Autonomie et reconfiguration des systèmes de systèmes tactiques

Marie Ludwig

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale SICMA

présentée par
Marie Ludwig

Préparée au Laboratoire Lab-STICC / ENSTA Bretagne,
sous contrat CIFRE chez
Thales Communications & Security

Autonomie et reconfiguration des systèmes de systèmes tactiques

Thèse soutenue le 24 octobre 2013
devant le jury composé de :

Bernard COULETTE
Professeur, IRIT - Université de Toulouse 2 / *Président du jury*

Antoine BEUGNARD
Professeur, Telecom Bretagne / *Rapporteur*

Vincent CHAPURLAT
Professeur, Ecole des Mines d'Alès / *Rapporteur*

Alain PLANTEC
Maître de Conférence, HDR, Lab-STICC - UBO / *Examinateur*

Jean-Philippe BABAU
Professeur, Lab-STICC - UBO / *Directeur de thèse*

Joël CHAMPEAU
Enseignant-Chercheur, Lab-STICC - ENSTA Bretagne / *Encadrant*

Nicolas FARCET
Docteur en informatique - Ingénieur, Thales Communications & Security / *Encadrant industriel*

Résumé

La complexité croissante des Systèmes de Systèmes et autres grandes fédérations d'acteurs pose de nouvelles problématiques de conception et de réalisation. Cette complexité, induite par des structures de management toujours plus sophistiquées et un cycle de vie long, doit être maîtrisée au plus tôt dans la conception des entreprises. Cette maîtrise permet à l'ensemble des intervenants au cours du cycle de vie d'une entreprise d'identifier ses points clés et de prendre confiance en sa capacité à atteindre ses objectifs. En particulier, il importe de savoir estimer les capacités de l'entreprise à s'adapter à des situations imprévues ou exceptionnelles afin d'assurer ses missions en toutes circonstances.

En réaction, de nouvelles démarches d'ingénierie émergent. Elles s'appuient sur la modélisation et la simulation de l'architecture de ces systèmes aux différents stades de leur développement et de leur fonctionnement.

Dans le cadre d'une de ces démarches nommée IDEA, nous avons enrichi le langage de description d'architecture avec des concepts et des mécanismes ayant pour but d'adresser l'adaptabilité et des capacités de reconfiguration des entreprises. Ces apports ont été expérimentés avec succès par prototypage et dans des contextes d'affaires industrielles.

Mots clés

Système de système ; Architecture ; Démarche d'Architecture ; Modélisation : Simulation ; Langage de Description d'Architecture ; Adaptabilité ; Reconfiguration

Abstract

As the complexity of large civilian and military Systems of Systems and system federations increases, new system architecture and engineering challenge emerge. This complexity is mainly due to intricate management structures and a long lifecycle, and needs to be mastered from the early stages of architecting. All engineering stakeholders need to identify the key aspects of the enterprise and gain confidence in its ability to fulfill its missions. To ensure that the enterprise is able to satisfy its objectives despite evolving situations, there is a need to focus on its capability to adapt through reconfiguration.

New engineering approaches emphasize architecture modelling and simulation to tackle the complexity of the enterprise in all stages of its lifecycle in a flexible and global way.

In the context of such an approach named IDEA, we updated the architecture description language to include concepts and mechanisms dedicated to the adaptability and reconfiguration of the enterprise. We also focused on ensuring model consistency. The results were experimented through prototyping and application on industrial affairs.

Keywords

System of system ; Architecture ; Architecting ; Modelling ; Simulation : Architecture Description Language ; Adaptability ; Reconfiguration

Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) – Pôle Communications, Architectures, Circuits et Systèmes (CACS)

Université de Bretagne Occidentale

6, avenue Le Gorgeu

29238 BREST Cedex 3

Thales Communications & Security - Customer Innovation Centre (CIC)

4 avenue des Louvresses

92622 GENNEVILLIERS Cedex

Remerciements

Je tiens avant tout à exprimer tous mes remerciements à Nicolas Farcet, Jean-Philippe Babau et Joël Champeau, pour m'avoir encadrée tout au long de cette thèse. Je remercie également Xavier Lagrenade, Arnaud Gauthier et Claire Amiraux de m'avoir accueillie au sein du CIC à Thales.

Je remercie également Benoît Baudry et Vincent Chapurlat, pour avoir accepté le rôle de rapporteurs malgré des délais ambitieux.

Merci à Rémi Courtel et Pascal Recchia, de l'équipe IDEA, pour le partage de leurs expériences lors de nos séances de travail communes, mais aussi pour l'excellente ambiance de travail qu'ils ont su maintenir.

Merci enfin tout spécialement à Sonia Dubois pour sa gentillesse et sa présence efficace dans tous les aspects de la vie en entreprise.

Table des matières

Table des illustrations

Chapitre 1. Introduction générale et positionnement

1.1. Présentation du domaine

1.1.1. Systèmes de Système, Entreprises et Command & Control

La notion de système de systèmes se retrouve dans différents contextes, en particulier les technologies de l'information et le domaine militaire. Dans le domaine des technologies de l'information, les « systèmes de systèmes d'information » tirent leur origine dans l'hétérogénéité (tant en termes de métiers que de technologies) des applications utilisées par les différentes entités d'une organisation. La maitrise de la complexité du système d'information global (le « système de systèmes ») devient alors un enjeu majeur pour l'organisation. Dans le domaine militaire, ce sont la complexité croissante des forces déployées (dispositifs interarmées et interalliés) et la recherche de l'exploitation en réseau des moyens de ces forces qui ont fait émerger la notion de système de systèmes. En France, cette notion est au cœur des préoccupations de la DGA (Délégation Générale pour l'Armement) depuis plusieurs années.

Quel que soit le contexte d'origine, la notion de système de systèmes adresse des assemblages de systèmes répondant à des caractéristiques spécifiques. En 2008, Luzeaux et Ruault proposent la définition suivante :

- **Système de systèmes (SdS) –** Un système de systèmes est un assemblage de systèmes pouvant potentiellement être acquis et/ou utilisés indépendamment, pour lequel le concepteur, l'acquéreur et/ou l'utilisateur cherche à maximiser la performance de la chaîne de valeur globale, à un instant donné et pour un ensemble d'assemblages envisageables [2].

Les critères de Maier, proposés en 1998, identifient un certain nombre de caractéristiques décrivant précisément les SdS [1] :

- **Indépendance opérationnelle** des systèmes composant le SdS : chacun d'entre eux peut être utilisé individuellement,
- **Indépendance managériale** de ces systèmes : les systèmes sont potentiellement issus de chaînes d'acquisition différentes, et maintenus par des organisations différentes,
- **Développement évolutionniste** du SdS : la composition du SdS n'est pas figée, et est sujette à évolution tout au long de son cycle de vie,
- **Comportements émergents** : les capacités du SdS dépassent la somme des capacités des systèmes qui le composent,
- **Distribution géographique** : les systèmes composant le SdS ne sont pas tous co-localisés,
- Nous prendrons également en considération un critère complémentaire de **finalités communes**, tel que proposé par la DGA en introduction à l'Ecole Systèmes de Systèmes de 2007. Ce critère introduit un objectif commun en vue duquel sont établies l'organisation et la coordination des composantes du SdS.

Dans le contexte industriel de cette thèse, nous nous intéresserons en particulier aux **SdS du domaine militaire**. Dans ce domaine, l'étude des SdS se positionne dans le cadre plus large du Command & Control (C2). Le C2 est un concept tirant son origine des analyses de la stratégie et des organisations militaires, et pour lequel nous considèrerons la définition suivante proposée par le Department of Defense (DoD) américain :

- **Command & Control (C2)** – *C2 is the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and Control functions are performed through the **C2 system**, an arrangement of personnel, equipment, communications, facilities and procedures employed by a commander in planning, directing, coordinating and controlling forces and operations in the accomplishment of the mission* [8].

L'un des points majeurs dans l'étude du C2 est l'optimisation du système assurant ses fonctions (« *C2 system* » dans la définition précédente), lequel peut éventuellement être un SdS. L'optimisation doit prendre en compte des aspects tels que la maîtrise du risque ou la résilience[1], dans des domaines aussi variés que l'informatique (réseaux…) et les sciences cognitives (comportements humains…). Dans [3], Bayne précise le besoin d'inclure dans la théorie du C2 une définition claire du système assurant les fonctions du C2, mais aussi de l'entité faisant l'objet de la gouvernance (les « *forces* » dans la définition précédente, pouvant également constituer un SdS). Cette définition doit permettre d'adresser de manière aussi formelle que possible l'ensemble des problématiques inhérentes à l'optimisation du système de C2, dans le but d'optimiser les capacités de l'ensemble (« *C2 system* » et « *forces* ») à remplir ses missions. C'est cet ensemble qui constitue alors l'entreprise (« *enterprise* ») au sens de Bayne :

- **Enterprise** – *The term "enterprise" refers to a quantifiable unit of organization (of arbitrary size), including its human and synthetic actors and information systems.* [3]

La définition précédente, très générique, ne restreint pas les entreprises au seul domaine militaire, ni aux seuls systèmes de systèmes : elle recouvre toute **fédération d'acteurs, aussi bien humains que synthétiques (logiciels, mécaniques…), collaborant pour une mission commune**. C'est dans ce sens que nous utiliserons dans toute cette thèse le terme d'**entreprise** pour désigner l'objet de l'étude, incluant en particulier les systèmes de systèmes tels que caractérisés au début de cette partie.

1.1.2. Architecture d'entreprise

Bayne définit la notion d'architecture d'une telle entreprise (« *architecture of enterprise* »[2]) comme ce qui permet à l'entreprise d'être viable, d'évoluer, et de maintenir ses capacités opérationnelles. L'architecture est alors définie par le biais de la spécification de la structure de l'entreprise, et des fonctions qu'elle réalise. Plus concrètement, nous entendrons sous la notion d'architecture d'entreprise la définition des différents **constituants de l'entreprise** (ex. acteurs, processus mis en œuvre, produits créés et échangés…) et de la **structure selon laquelle ils s'articulent** (ex. liens entre les acteurs et les processus…)

Dans le domaine militaire, nous considérons à titre d'exemple une entreprise constituée par un ensemble de forces déployées sur un théâtre d'opérations. Cet ensemble peut être de volume variable selon les objectifs de l'étude, et inclut à la fois le personnel et le matériel (véhicules, équipements de communication, systèmes de communication…). L'architecture de cet ensemble de forces comprend notamment les structures techniques permettant les échanges entre les éléments

[1] Capacité d'un système à retrouver un état stable suite à une perturbation.
[2] A ne pas confondre avec le terme anglophone « *enterprise architecture* », qui concerne pour sa part l'urbanisation des systèmes d'informations.

composant la force (radio, réseaux informatiques…), mais aussi l'organisation hiérarchique des acteurs, et la doctrine qui dirige leurs actions.

Les objectifs de l'étude de l'architecture d'une telle entreprise peuvent être de natures diverses, selon les intérêts de l'entité en charge de la réaliser. Par exemple :

- Concevoir le système de communication nécessaire à assurer et optimiser l'échange et le partage d'information entre les différents acteurs,
- Evaluer l'impact du remplacement d'un système par un autre au sein de l'entreprise,
- Etudier et comparer différentes procédures à mettre en œuvre dans des situations données.

Les deux exemples suivants illustrent brièvement des problématiques que peuvent soulever ces études en termes d'architecture d'entreprise.

Exemple 1. Un convoi militaire, en charge d'acheminer du matériel entre deux points, doit être capable de se réorganiser pour faire face à une attaque ou tout autre événement imprévu. L'architecture du convoi en termes de procédures et d'organisation (avant et après l'événement) doit avoir été prévue pour supporter cette réorganisation, de manière à pouvoir dispenser aux équipages l'entraînement nécessaire au bon déroulement de la réorganisation en cours d'opération.

Cet exemple sera utilisé tout au long de cette thèse pour illustrer les propositions : il sera présenté plus en détail en 2.1.

Exemple 2. Dans le cadre de la logistique de théâtre, il faut prévoir des procédures à mettre en œuvre pour évacuer vers l'arrière un matériel endommagé dans une zone de combat. Cette évacuation n'est pas forcément réalisée par des personnels et matériels spécifiques, mais peut être gérée par le personnel effectuant la mission nominale. L'identification des critères de sélection du personnel en charge de l'évacuation est d'une importance capitale, dans la mesure où l'impact sur la mission est important : le personnel sélectionné devra interrompre ses activités en cours pour prendre en charge l'évacuation

L'architecture de l'entreprise doit permettre de prendre en compte ces critères pour choisir les acteurs les plus appropriés, puis de constituer une collaboration opportune pour l'évacuation. Cette collaboration impacte les procédures et organisations humaines, mais également les systèmes d'information : en effet, les acteurs sélectionnés pour réaliser l'évacuation devront disposer d'informations sans rapport avec leur mission nominale (ex. lieux de rendez-vous), et interagir avec de nouveaux partenaires (ex. les unités logistiques).

Cet exemple est issu d'une affaire industrielle sur laquelle ont été appliqués les résultats de cette thèse : il sera repris dans la partie 3.2.3.2 pour illustrer les travaux effectués.

1.1.3. Modélisation et simulation en support aux activités d'architecture

La complexité inhérente aux entreprises telles que définies précédemment soulève d'importants défis tout au long de leur cycle de vie. En particulier, nous nous intéresserons ici à la démarche d'architecture (« *architecting* »), en amont de leur phase de conception. Dans le monde militaire, par exemple, la démarche d'architecture intervient avant le lancement des programmes de conception des composantes de l'entreprise (ex. véhicules, systèmes d'information…).

Du point de vue du cycle global de conception de l'entreprise, les principaux enjeux de la démarche d'architecture sont les suivants :

- Maîtriser la complexité de l'entreprise et appréhender ses points clés,
    - Ex. Identifier les systèmes critiques
- Prendre confiance dans les capacités de l'entreprise à assumer ses missions et remplir ses objectifs,
    - Ex. Optimiser la disponibilité des systèmes critiques
- Prendre confiance dans les capacités de l'entreprise à se maintenir dans un environnement difficilement prévisible (voire hostile dans le domaine militaire).
    - Ex. Prévoir les modalités de résilience de l'entreprise (face aux réorientations de missions, aux évènements imprévus…)

Les approches d'ingénierie système dirigée par les modèles (MBSE – Model-Based Systems Engineering) [9], principalement mises en œuvre dans le monde des technologies de l'information, préconisent l'utilisation d'un modèle d'architecture. Ce modèle est partagé et compris par l'ensemble des parties prenantes (ex. architectes, utilisateurs finaux…), et peut être exploité (voire raffiné) tout au long du cycle de vie de l'entreprise. Il peut ainsi servir en tant que référence pendant les travaux d'ingénierie ou plus tard au cours de l'exploitation de l'entreprise réalisée.

Pour assurer au mieux la pertinence du modèle (et donc de l'entreprise elle-même) vis-à-vis du domaine considéré, il est important d'élaborer celui-ci en coopération étroite avec les experts du domaine (opérationnels). Cette coopération permet également une meilleure compréhension du besoin. La principale difficulté rencontrée lors de l'implication d'experts opérationnels dans une démarche de modélisation est généralement l'adaptation au formalisme : dans la majeure partie des cas les opérationnels sont peu (voire pas du tout) familiers du domaine de la modélisation et des langages couramment utilisés dans le monde informatique. Afin de capturer au mieux leur expertise du domaine, il faut alors leur présenter le modèle d'une manière parlante et expressive, proche autant que possible de leurs références usuelles (langage, symbologie…). Les approches basées sur la simulation, présentant l'entreprise au plus proche de la réalité, sont ainsi couramment mises en œuvre dans une telle optique.

Nous nous positionnons pour cette thèse dans le cadre une démarche d'architecture d'entreprise exploitant à la fois un **modèle d'architecture** constituant une référence pour l'ensemble de la conception de l'entreprise, et une **simulation réaliste** permettant d'impliquer des **experts opérationnels**. Pour garantir la cohérence entre modélisation d'architecture et simulation, nous recherchons à nous appuyer sur **un même modèle** pour ces deux activités.

En disposant d'un tel modèle, la démarche peut alors s'articuler autour de boucles modélisation / simulation pour une modélisation incrémentale de l'architecture. Pour que l'implication des opérationnels soit la plus interactive possible, ces boucles doivent pouvoir se faire sur une durée très courte, de l'ordre de quelques heures à une journée. Au cours d'une même réunion, il est ainsi possible de confronter une version initiale du modèle ou d'une partie du modèle à l'expertise des opérationnels, puis d'effectuer avec eux les modifications qu'ils jugent nécessaires. Pour désigner une telle approche de l'élaboration du modèle, nous emploierons le terme de **prototypage rapide**.

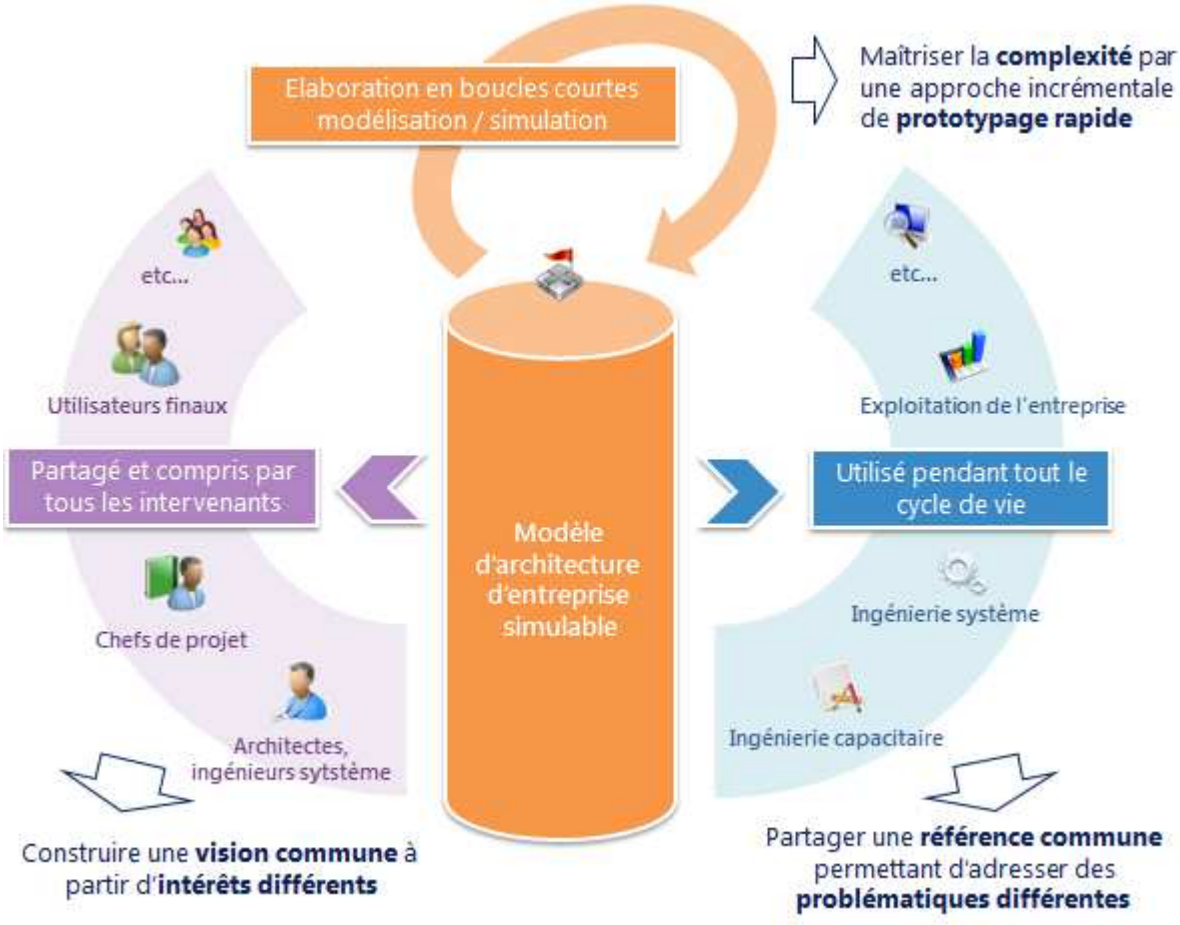


Figure 1 – Les apports d'une démarche d'architecture d'entreprise basée sur un modèle simulable

1.2. Etat de l'art

1.2.1. Modélisation et simulation d'architectures d'entreprise

1.2.1.1. Généralités sur la modélisation et la métamodélisation

Indépendamment du domaine concerné par les activités de modélisation, l'Object Management Group (OMG) définit quatre niveaux de modèles (*4-layer model* – [10]), représentant quatre niveaux d'abstraction :

- Le **niveau M0** correspond au **système réel**, faisant l'objet de la modélisation ;
- Le **niveau M1** correspond au **modèle** du système réel, c'est à dire une représentation conceptuelle de ce système, basée sur un langage de modélisation ;
- Le **niveau M2** correspond au **métamodèle**, c'est à dire à la définition de ce langage de modélisation ;
    - En termes de standards OMG, le niveau M2 correspond à l'UML (*Unified Modeling Language* [12]), permettant de réaliser des modèles utilisateur de niveau M1.
- Le **niveau M3** correspond au **méta-métamodèle**, c'est à dire à la définition du langage dans lequel est exprimé le métamodèle. Par principe, le méta-métamodèle doit pouvoir se définir lui-même.

- En termes de standards OMG, le niveau M3 correspond au MOF (*Meta-Object Facility*).

1.2.1.2. Modélisation d'architectures d'entreprise dans le domaine du Command & Control

Dans l'industrie de la Défense, les projets de modélisation utilisent couramment le modèle JC3IEDM[3] de l'OTAN [29]. Celui-ci a pour objectif de permettre l'interopérabilité entre les systèmes d'information militaires des différentes nations, en proposant un modèle commun pour décrire les organisations militaires interarmées. Il s'agit cependant d'un modèle de données, visant à être exploité par des systèmes informatiques, et non d'un modèle ayant vocation à adresser des problématiques de modélisation d'architecture d'entreprise. Par exemple, il s'attache principalement à la description de matériels existants et ne supporte ainsi pas la prise en compte de systèmes futurs.

Dans le monde du Command & Control, les travaux de référence concernant la modélisation d'architectures d'entreprise sont plutôt ceux de Bayne [3][4][5]. Ces travaux proposent un formalisme de modélisation et des principes mathématiques permettant d'optimiser le système modélisé.

Les modèles de Bayne sont basés sur la notion d'''unités de productions'' ou **VPU**s (*Value Producing Unit*). Chaque VPU composant l'entreprise effectue ses tâches en communiquant selon deux axes :

- L'axe hiérarchique (vertical sur la Figure 2) permet au VPU de transmettre ses rapports à un supérieur et ses ordres à un subalterne.

- L'axe logistique (horizontal sur la Figure 2) permet au VPU de transmettre ses requêtes à un fournisseur et ses produits à un client.

Les travaux de Bayne sont essentiellement exploités et utilisés dans le monde de la recherche académique et de l'analyse opérationnelle (ex. par les organisations gouvernementales). Dans l'industrie, la modélisation d'architecture d'entreprise est essentiellement mise en œuvre en utilisant le langage **UML** (Unified Modeling Language) [11][12][20] ou les **cadres d'architecture** (*Architecture Frameworks* ou **AF**) [6][7].

[3] *Joint Consultation, Command and Control Information Exchange Data Model* - STANAG 55254

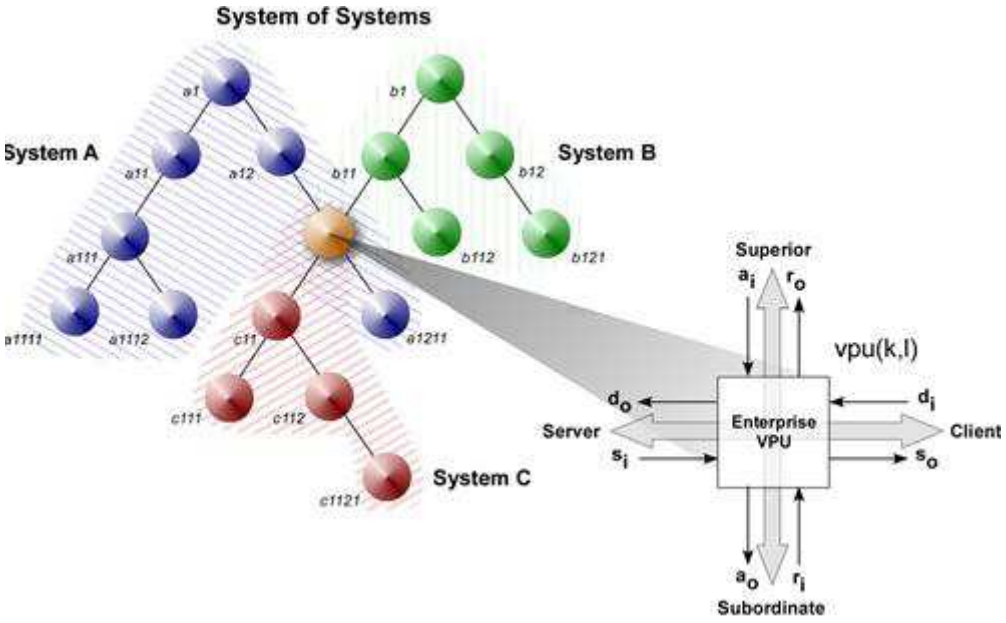


Figure 2 - Le concept de VPU [3]

UML est le langage de modélisation objet standard de l'OMG, développé pour le domaine logiciel. Il propose des concepts adaptés pour définir les éléments du modèle (ex. classe, acteur…) et des types de diagrammes permettant d'orchestrer les relations entre les éléments. UML 2.3 définit 13 types de diagrammes, répartis en trois grands groupes [12] :

- Les diagrammes structurels (ou statiques), permettant de décrire la structure du système étudié (ex. diagramme de classes),
- Les diagrammes comportementaux (ou dynamiques), permettant de décrire le comportement du système étudié (ex. diagramme d'activité),
- Parmi les diagrammes comportementaux, on distingue les diagrammes d'interaction, permettant de décrire la manière donc collaborent les différents constituants du système étudié (ex. diagramme de séquence).

Pour adapter UML au domaine plus spécifique de l'ingénierie système, l'OMG a adopté en 2006 le langage SysML (Systems Modeling Language) [13]. SysML propose une sémantique plus riche qu'UML et introduit de nouveaux types diagrammes (en particulier, le diagramme des exigences), permettant ainsi d'adresser des problématiques plus larges que le génie logiciel.

Les **cadres d'architecture** sont des standards initialement définis pour la gouvernance des grands programmes industriels du domaine de la Défense, afin de permettre aux agences étatiques (ex. Department of Defense américain, DGA française) de partager avec les industriels une vision commune du système à réaliser. Les principaux cadres d'architecture sont le DoDAF (Department of Defense Architecture Framework) [6] et le NAF (NATO Architecture Framework) [7]. Ils définissent un ensemble de *produits* spécifiant chacun la description d'un aspect du système à étudier : par exemple, le produit NAF NCV-1 « *Capability Taxonomy* » spécifie la description des différentes capacités de l'entreprise. Les produits adressent principalement l'architecture de l'entreprise selon trois niveaux de granularité ou *vues*, adressant les intérêts de différentes parties prenantes vis-à-vis du système :

- Opérationnel (*Operational views*), permettant de décrire les aspects métier de l'entreprise,

- Système (*System views*), permettant de décrire le système (ou système de système) mis en œuvre pour exercer les aspects métier de l'entreprise,
- Technique (*Technical views*), permettant de décrire l'implémentation technique (ex. standards, protocoles…) du système.

Contrairement à UML, les cadres d'architecture ne spécifient pas les modalités graphiques de réalisation de chacun des produits (ex. symbologie). La plupart du temps, les éditeurs logiciels proposent des représentations graphiques basées sur celles d'UML.

1.2.1.3. Simulation de modèles d'architecture d'entreprise

Y compris dans le monde militaire, la notion de simulation d'architecture d'entreprise diffère de celle de simulation opérationnelle. Si la première a pour objectif d'étudier le système constitué par la force (incluant son matériel, ses échanges etc…), la seconde permet d'évaluer la pertinence du choix des unités militaires impliquées par rapport au terrain et au comportement de l'ennemi.

Pour les architectures d'entreprises, potentiellement complexes à comprendre et donc à simuler, les approches MBSE représentent un atout indéniable. Un certain nombre de travaux de recherche s'intéressent depuis plusieurs années au lien entre la modélisation d'architecture et la simulation [33][34][35]. En particulier, certains se penchent sur la problématique des **modèles d'architecture simulables**, visant à présenter les aspects dynamiques d'une entreprise de manière réaliste, à la fois pour permettre une meilleure compréhension de ces aspects et pour pouvoir réaliser des évaluations et des mesures quantitatives [14][17][19][36].

L'agence canadienne de Recherche et Développement pour la Défense (RDDC) considère les modèles d'architecture simulables (« *executable architectures* ») comme un point clé à atteindre pour faire le lien entre la démarche d'architecture mise en œuvre pour concevoir une entreprise, et la simulation permettant d'évaluer l'architecture définie [17]. Elle définit un modèle d'architecture simulable comme un « modèle dynamique du comportement d'un système, où les éléments d'architecture sont identifiés individuellement, exploités de manière cohérente et organisé d'une manière permettant leur simulation »[4]. En 2008, elle a publié un état de l'art visant à identifier les différents standards MBSE actuels et les évaluer en fonction de leur capacité à supporter la simulation. La conclusion de cet état de l'art préconise l'utilisation de plusieurs standards de modélisation, parmi lesquels le profil UML UPDM (UML Profile for DoDAF and MoDAF[5]) [30] et le langage xUML (executable UML) [31], et le standard HLA (High-Level Architecture) [32] pour fédérer plusieurs outils de simulation.

De manière générale, les travaux concernant la simulation des modèles d'architecture d'entreprise font apparaître deux types de démarches :

- Les démarches issues du monde de la modélisation, dont l'objectif est généralement de rendre simulable un modèle d'architecture réalisé en utilisant un standard,
- Les démarches issues du monde de la simulation, qui cherchent généralement à établir un lien entre une méthode ou un outil de simulation existant et l'architecture globale de l'entreprise.

[4] "*The dynamic model of the behavior of a system where the architecture elements are uniquely identified, consistently used and structured in a way to enable their simulation*" [17]
[5] MoDAF : *Ministry of Defense Architecture Framework*. Il s'agit du cadre d'architecture du ministère de la Défense britannique, fortement inspiré du DoDAF américain.

Parmi les démarches issues du monde de la modélisation, l'une des plus abouties et la méthodologie présentée par Pawlowski et al. (MITRE Corp.), permettant l'import de produits DoDAF dans un formalisme exécutable [17]. Leur approche est basée sur la transformation de produits DoDAF vers des outils de simulation, mettant en œuvre différents métamodèles. La méthodologie définie a été concrétisée par le développement d'un logiciel de transformation, ICAMS (*Integrated C4ISR*[6] *Analysis and Management System*), permettant de générer les données nécessaires à un outil de simulation à partir de produits DoDAF réalisés avec l'outil Enterprise Architect, et de maintenir la traçabilité entre les éléments du modèle de simulation et ceux du produit.

Le standard HLA [32] recommandé par le RDDC est au cœur de la majorité des démarches issues du monde de la simulation. Il permet l'interopérabilité technique entre différents outils de simulation, mais n'adresse pas la problématique d'ensemble de la cohérence au niveau conceptuel entre les éléments simulés. Dans [35], Kewley et al. présentent une approche globale basée sur le langage UML visant à assurer cette cohérence d'ensemble.

1.2.2. Définition du langage

Pour appréhender plus précisément les principes pouvant guider la conception d'un langage permettant la modélisation et la simulation d'architectures d'entreprise, il est également intéressant de prendre pour référence des travaux issus de domaines différents. Ces travaux nous permettront dans la suite du document de construire des réponses concrètes à certaines problématiques particulières rencontrées au cours de l'étude.

1.2.2.1. Méthodes de définition d'un langage de modélisation

Dans le cadre des standards OMG, la principale méthode de définition d'un langage de modélisation passe par la définition d'un **profil UML**. Un profil permet de personnaliser UML afin de l'adapter à un domaine particulier, par la constitution d'extensions au langage initial (ajout de stéréotypes, de contraintes…). Les profils ne permettent pas de modifier UML lui-même, et héritent donc de sa sémantique (concepts de classe, d'objet…), de ses notations graphiques, et de l'obligation d'être conformes à son métamodèle.

Pour notre étude, il importe que le langage soit accessible aux opérationnels pour optimiser l'approche par prototypage rapide : la sémantique UML, dédiée à un public d'ingénieurs et mettant en œuvre des concepts liés au domaine des technologies de l'information, n'est pas adaptée. Au contraire, il importe pour nous de mettre l'accent sur une **sémantique métier** afin que le langage soit le plus expressif possible pour les utilisateurs. Plutôt que l'approche par profil UML, nous avons choisi une approche basée sur un **Domain Specific Language (DSL)** : à l'opposé d'un langage générique, permettant d'adresser un grand nombre de domaines d'applications, un DSL est un langage dont les spécifications (syntaxe, représentation…) sont propres à un domaine d'application particulier. Les DSL peuvent aussi bien être des langages graphiques que textuels (ex. HTML).

Les principaux outils permettant de développer un DSL sont à l'heure actuelle les outils DSL Tools de Microsoft, présentés sous la forme d'une extension de l'environnement de développement Visual Studio[7], et l'Eclipse Modeling Foundation (EMF) intégré à Eclipse. Développé par l'IRISA (Institut de

[6] C4ISR : *Computerized Command, Control, Communications, Intelligence, Surveillance, Reconnaissance*. Sigle communément utilisé pour évoquer un ensemble de fonctions militaires et les systèmes qu'elles emploient.
[7] Depuis la version 2008

Recherche en Informatique et Systèmes Aléatoires) de Rennes, Kermeta est également un environnement de métamodélisation, lui-même défini par un DSL [18]. Kermeta exploite EMF, et est intégré à Eclipse.

1.2.2.2. Différents types de langage

Même si les DSL n'ont pas tous vocation à concerner le domaine informatique, la définition de leur structure de base peut s'appuyer de celle des langages de programmation.

Dans les **langages de programmation objet à classe**, une classe définit les caractéristiques communes (attributs et opérations) à un ensemble d'objet de même nature. Les objets décrits par une classe sont nommés les instances de cette classe. Dans le monde de la modélisation, l'UML permet de décrire des systèmes implémentant ce paradigme. Dans le métamodèle UML 2.4.1, l'élément *InstanceSpecification* représente une instance dans le système d'étude. C'est un élément *Classifier* qui représente l'entité dont l'*InstanceSpecification* est une instance. Une vue d'ensemble de ces concepts dans le package *Instances* est présentée sur la Figure 3 telle que définie dans le document de spécification d'UML 2.4.1 [20].

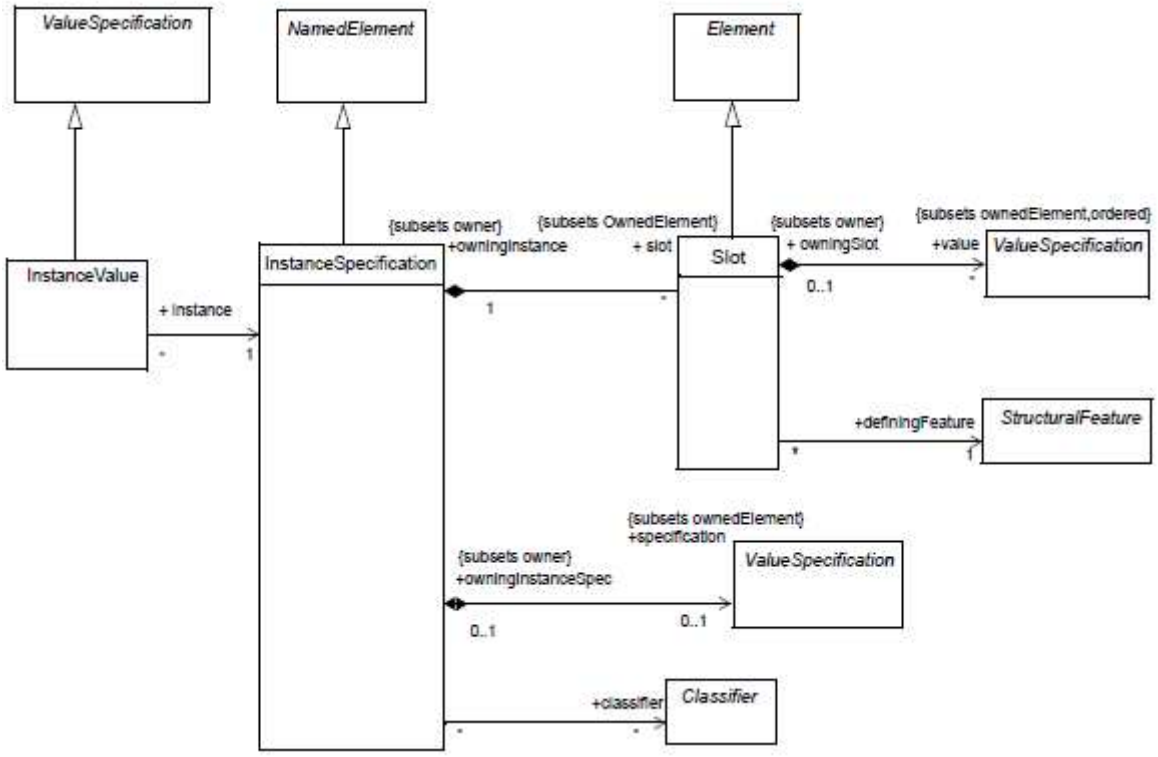


Figure 3 - Les éléments définis dans le package *Instances* d'UML 2.4.1 (issu de [20])

Le concept *InstanceSpecification* présente les propriétés suivantes :

- Le *Classifier* représente l'entité dont l'*InstanceSpecification* est une instance. Par exemple, une *InstanceSpecification* dont le *Classifier* est une classe représente un objet de cette classe. On notera qu'une *InstanceSpecification* peut être liée à plusieurs *Classifiers*, autorisant ainsi l'héritage multiple.
- Les *Slot* donnent les valeurs définies par l'*InstanceSpecification* pour les *StructuralFeature* de son *Classifier*.

- La *Specification* définit le mode d'utilisation de l'*InstanceSpecification* (la construire, en dériver…)

L'association entre *InstanceValue* et *InstanceSpecification* traduit le fait qu'une valeur représentée par une *ValueSpecification* peut être une instance.

Les langages à classe ne sont pas les seuls types de langages de programmation objet : en particulier, la classification proposée dans [42] contient également la notion de **langage à prototype**. Cette notion introduit le concept de prototype, comme un objet servant de référence dans le processus de création d'autres objets dénotant le même concept[8]. L'un des mécanismes les plus caractéristiques des langages à prototype est le clonage. Le clonage d'un objet consiste à effectuer une copie de celui-ci, afin d'obtenir un nouvel objet possédant le même comportement et les mêmes propriétés dotées des mêmes valeurs. Les deux objets pourront par la suite évoluer indépendamment : par exemple, la mise à jour de la valeur d'une propriété sur un des clones n'aura pas d'impact sur l'autre.

1.3. Proposition et positionnement

1.3.1. Cadre et périmètre des modèles

1.3.1.1. Périmètre et niveau d'abstraction

Les entreprises concernées par l'approche d'architecture d'entreprise dans laquelle nous nous positionnons peuvent *a priori* relever de métiers très différents, comme par exemple la logistique militaire ou la gestion du trafic aérien. Pour que le langage de modélisation soit applicable dans le cadre de tous ces métiers, il importe donc qu'il reste suffisamment générique. Il ne doit pas comprendre de termes qui soient propres à un domaine d'application spécifique, mais être basé sur des concepts génériques, desquels les experts des différents métiers peuvent dériver la description de leur entreprise.

Les langages de modélisation UML ou SysML sont basés sur des concepts très génériques, choisis pour pouvoir être appliqués ou adaptés à un grand nombre de domaines. Ces langages sont généralement familiers aux ingénieurs (en particulier dans le domaine des technologies de l'information), et sont pour cette raison souvent utilisés en support aux différentes phases d'ingénierie. Ces langages ont toutefois été initialement conçus pour la modélisation de systèmes de niveau de granularité plus fine qu'un Système de Systèmes ou une entreprise, en l'occurrence principalement un logiciel pour UML, et une plateforme (ex. un véhicule) ou un équipement pour SysML. Ils restent de plus peu parlants pour les experts métier (ex. membres des forces armées pour les entreprises concernant le domaine militaire), et nécessitent un important effort d'apprentissage et de traduction de leur part.

Par exemple, la notion de *capacité* est primordiale dans le domaine de la Défense et de la sécurité. Cette notion n'est présente ni dans UML ni dans SysML, aussi la modélisation d'une *capacité* dans l'un de ces langages nécessite un choix d'interprétation. En UML, on pourrait ainsi considérer une

[8] Pour plus d'information sur la notion de prototype ou les principes des langages à prototype, on pourra se référer à [42] et à ses références.

capacité comme un *Use Case*, dans le sens où elle représente une attente vis-à-vis de l'entreprise (ou de l'un de ses constituants). Mais on pourrait également la modéliser comme un *Package*, pour traduire le fait que sa réalisation est un objectif pour un certain sous-ensemble de l'entreprise.

La nécessité de faire de tels choix d'interprétation, associée à l'importance de l'effort d'apprentissage, est un obstacle à la capture de la connaissance des experts opérationnels. L'utilisation de ces langages de modélisation se révèle ainsi peu adaptée à une approche d'architecture d'entreprise visant à impliquer au plus près les experts métier. Les modèles utilisés pour une telle approche doivent mettre en œuvre des concepts et une sémantique métier, qui soit parlante non seulement aux ingénieurs mais aussi aux experts opérationnels ou aux clients.

Les modèles présentés par Bayne donnent une représentation formelle de l'organisation et des activités génériques d'une entreprise afin de permettre l'optimisation de ses processus de C2, et donc son efficacité globale [3][4][5]. Ces modèles définissent un formalisme mathématique, inspiré du domaine de l'automatique (contrôle-commande) et adapté au domaine des entreprises. Ils permettent de caractériser certains aspects de l'entreprise pour son optimisation, mais n'ont pas vocation à représenter son architecture.

Par exemple en ce qui concerne les aspects fonctionnels, les modèles de Bayne identifient au sein d'un VPU huit rôles entièrement caractérisés en termes de structure et de responsabilité [3]. Non extensibles, les modèles ne permettent pas l'introduction de rôles supplémentaires qui seraient propres au métier de l'entreprise étudiée et nécessaires pour l'étude de son architecture et l'expressivité du modèle (ex. le rôle de Contrôleur Aérien dans le métier de la gestion du trafic aérien).

N'ayant pas pour objectif de décrire l'architecture d'une entreprise mais de l'abstraire pour permettre son évaluation par les mathématiques, les modèles de Bayne ne sont pas adaptés à la représentation et l'évaluation de variantes d'architectures différentes dans le cadre d'une démarche d'architecture d'entreprise en phase amont de conception. De plus, ces modèles restent trop théoriques pour être parlants et représentatifs aux yeux d'experts opérationnels.

Les cadres d'architecture (AF) ont été spécifiés à la demande d'acteurs du domaine militaire (US Department of Defense, OTAN…) mais sont également applicables dans d'autres domaines. Par exemple, Dixson et Genik présentent dans [16] un retour d'expérience de la mise en œuvre d'un cadre d'architecture[9] dans le cadre de la planification des aspects sécurité des Jeux Olympiques de Vancouver en 2010. L'AF leur a permis de modéliser aussi bien des constituants relevant du domaine de la sécurité civile (pompiers, services de santé…) que de l'organisation même de l'évènement (comité d'organisation, supervision du site…), en coopération étroite avec des experts de ces domaines.

Le niveau d'abstraction et de généricité des AFs semble ainsi bien adapté à l'approche considérée dans le cadre de la présente thèse. Cependant, ils ont pour principal objectif de permettre la conciliation des points de vue des différents intervenants de la conception de l'entreprise, y compris dans un but documentaire. Ils incluent pour ce faire des informations complémentaires à l'architecture, comme par exemple des vues d'artiste (ex. le produit NAF Operational View 1 (NOV-1)) ou des plannings de projets (ex. le produit NAF Capability View 3 (NCV-3)).

[9] En l'occurrence le DNDAF, *Department of National Defence Architecture Framework*, développé par le gouvernement Canadien sur la base du DoDAF des Etats-Unis

Dans le cadre d'une approche d'architecture par prototypage rapide (telle que présentée en 1.1.3), nous estimons qu'il est nécessaire de réduire le périmètre couvert par les modèles par rapport à celui de l'ensemble des vues des cadres d'architecture. Cela permettra de cibler les aspects clés de l'architecture, et d'éviter de proposer des points de vue qui ne seraient pas nécessaires à ce stade de la conception de l'entreprise.

En s'inspirant du niveau d'abstraction des AF, le périmètre des modèles sera concentré sur les aspects suivants :

- **Organisation** – Les entités (systèmes et acteurs humains), composant l'entreprise, ainsi que leurs liens de structure : commandement, décomposition en termes de niveau de granularité (contenu/contenant, groupe/membre de groupe), etc...
- **Comportement** – Les activités que les systèmes et acteurs qui composent l'entreprise déroulent pour mettre en œuvre leurs capacités et atteindre leurs objectifs (ex. Processus métiers BMPN [37]).
- **Interactions** – Les modalités d'échanges entre les entités composant l'entreprise : flux d'information, consommation et fourniture de services…

Les constituants d'architecture seront considérés à des niveaux de granularité allant de l'**opérationnel** (ex. l'ensemble des forces déployées sur un théâtre d'opérations) au **système** (ex. un véhicule ou un équipement). A ce stade de la conception de l'entreprise, nous ne chercherons pas à représenter les constituants de niveau technique (ex. les composants d'un senseur), de granularité trop précise par rapport aux problématiques adressées.

1.3.1.2. Exécution d'un modèle

Dans [17], L'agence canadienne de Recherche et Développement pour la Défense recommande pour l'élaboration d'architectures simulables l'utilisation d'une combinaison de plusieurs standards de modélisation, et de la norme HLA pour l'interopérabilité des simulations (cf. 1.2.1.3). Les outils de simulation à mettre en œuvre sont à sélectionner au cas par cas selon les objectifs de l'étude, de même que les standards de modélisation. Une fois que l'architecture est modélisée, cette approche requiert alors une transformation de modèle vers les outils de simulation.

Si l'approche ainsi identifiée est adaptée à l'évaluation par la simulation d'une architecture système, elle parait ainsi complexe à mettre en œuvre dans le cadre d'une approche de prototypage rapide. En effet, les phases de sélection des outils et standards et de transformation de modèle semblent incompatibles avec la mise en œuvre de boucles de modélisation / simulation d'une durée de l'ordre de quelques heures à une journée.

Par ailleurs, les simulations ainsi mises en œuvre sont dédiées à l'étude de certains aspects de l'entreprise (ex. processus métiers pour les outils BPMN, flux pour SIMUL8…), de manière indépendante du reste de l'architecture. Il est possible de conserver une traçabilité entre les éléments du modèle de simulation et ceux du modèle d'architecture [19], mais les simulations ne prennent pas en compte l'ensemble de l'architecture modélisée. Dans l'exemple de l'évacuation de matériel évoqué en 1.1.2, ces simulations ne permettent pas d'évaluer différents critères de sélection du personnel à charger de l'évacuation, les processus ne pouvant accéder aux informations des acteurs (leurs missions, leur charge...). Plus globalement, ces simulations ne permettent pas de visualiser le comportement de l'entreprise dans son ensemble comme attendu dans notre cadre d'étude.

Pour permettre des boucles courtes de modélisation / simulation, nous cherchons la possibilité de simuler directement l'architecture dans son ensemble sans mettre en œuvre de transformation de modèle. Plutôt que de projeter la sémantique du langage de description d'architecture vers une autre sémantique dédiée à la simulation, nous pensons en effet que l'ajout de la sémantique d'exécution directement au niveau du langage est plus adapté pour le prototypage rapide. Cette approche est similaire à celle adoptée par le langage de métamodélisation exécutable Kermeta (cf. 1.2.1.3).

Bien que les modèles issus des cadres d'architecture n'aient pas été conçus pour être simulés, certains outils implémentant ces formalismes proposent l'animation de produits dynamiques (généralement processus). Par exemple, le module d'extension Simulator pour l'outil de modélisation System Architect (IBM) supporte la simulation des processus BPMN modélisés dans les vues NAF NOV-5 (*Operational View 5 – Operational Activity Model*). Ces simulations permettent de visualiser de manière dynamique le comportement de l'entreprise et/ou de ses constituants, et de présenter rapidement les comportements modélisés de manière interactive et intuitive pour l'ensemble des intervenants de l'architecture. Elles sont généralement réalisées sans transformation de modèle, en s'appuyant sur le formalisme choisi par l'outilleur pour représenter les aspects dynamiques de l'entreprise[10]. Ces outils ne proposent pas de re-définition du périmètre de la simulation pour prendre en compte l'ensemble de l'architecture modélisés. Les informations considérées pour la simulation sont les mêmes que dans le cas d'un processus modélisé de manière indépendante dans un outil tiers, à savoir celles des vues représentant les aspects dynamiques de l'entreprise (ex. NOV-5).

L'obstacle principal à la prise en compte par ces simulations de l'ensemble de l'architecture provient de la vision éclatée de l'architecture proposée par les AFs et de leur manque de métamodèle défini de manière claire et cohérente. Les nombreuses tables de mapping entre les différentes vues tentent de maintenir une certaine cohérence dans l'architecture, mais introduisent des difficultés d'interprétation pour la simulation.

Pour permettre des boucles modélisation / simulation aussi courtes que possibles, il apparaît intéressant que les modèles permettent directement de simuler l'architecture dans son ensemble, sans qu'il y ait besoin de transformation de modèle. La simulation mise en œuvre est ainsi une simulation métier, par opposition à une simulation générique comme les approches illustrées précédemment pour la simulation des processus dans les AFs.

Pour ce faire, nous nous appuierons sur les points suivants :

- Intégrer la **modélisation des comportements** de l'entreprise sous forme de processus au sein du modèle d'architecture ;
- Permettre l'exécution de ces processus **sans transformation** du modèle d'architecture, en prenant en compte le contexte d'exécution des processus dans l'entreprise (ressources, disponibilité des fournisseurs de services…) ;

[10] Si les AFs définissent un contenu pour les vues représentant les aspects dynamiques d'une entreprise, ils n'imposent pas de formalisme de modélisation. Le choix de ce formalisme est donc laissé aux outilleurs. Par exemple, pour les vues NAF NOV-5, l'outil System Architect supporte deux notations différentes : BPMN et IDEF (ICAM (Integrated Computer-Aided Manufacturing) DEFinition).

- Lors de l'exécution, permettre aux processus d'accéder à l'**ensemble du modèle d'architecture** et d'effectuer des modifications sur ce modèle selon des modalités à définir.

Nous utiliserons le terme **modèle d'architecture exécutable** pour mentionner les modèles d'architecture ainsi considérés.

1.3.1.3. Expression de l'adaptabilité de l'entreprise

La caractéristique de développement évolutionniste identifié par Maier pour les systèmes de systèmes peut être étendue à l'ensemble des entreprises considérées dans le cadre de notre étude. En particulier dans le domaine militaire, elles ont pour vocation d'évoluer dans des environnements changeants et imprévisibles, et doivent ainsi pour pouvoir y faire face être capables d'adaptabilité. Comme déjà évoqué précédemment, la prise de confiance en cette capacité d'adaptabilité est l'un des enjeux majeurs de la démarche d'architecture. Dans une approche orientée modèle, il importe que le modèle permette de **représenter et d'évaluer l'adaptabilité de l'entreprise**.

Bien qu'il s'agisse d'un objectif primordial pour le modèle, il est indispensable que la représentation de l'adaptabilité de l'entreprise ne vienne pas à l'encontre des principes déterminants de la démarche retenue. Dans cette optique, nous nous attacherons à veiller tout particulièrement aux points suivants :

- La représentation de l'adaptabilité doit induire **le moins de complexité possible** dans les modèles, afin d'être accessible aux experts opérationnels et pouvoir être mise en œuvre rapidement dans une optique de prototypage rapide. En particulier, il importe que les moyens d'expression de l'adaptabilité restent **optionnels**, afin de permettre aux exploitants du modèle d'adresser la problématique au moment opportun, et non de les contraindre à prendre en compte l'adaptabilité quel que soit l'objectif de leurs travaux.
- L'**exécutabilité** du modèle est au cœur de la démarche d'évaluation de l'architecture : l'expression de l'adaptabilité, en particulier, doit pouvoir être évaluée par le biais d'une telle simulation.

1.3.2. Présentation de l'approche

En support à la création de modèles conformes aux principes présentés dans les parties précédentes, le projet dans lequel s'inscrit cette thèse a pour objectif la définition d'un **langage de description d'architecture d'entreprise**. Le principe général de ce langage est de supporter la définition et l'évaluation de différentes architectures candidates par la simulation, dans le cadre d'une démarche de prototypage rapide.

Le but recherché pour cette thèse est l'intégration de mécanismes permettant au langage d'**exprimer l'adaptabilité** de l'architecture d'entreprise, c'est à dire les éléments qui lui permettent de s'adapter aux différentes contraintes auxquelles elle est soumise (ex. environnement…).

1.3.2.1. Objectifs

Dans le cadre de cette approche, nous avons identifié un ensemble d'objectifs représentant les points essentiels pour que le langage et les modèles puissent remplir efficacement leur rôle. Ces objectifs ont pour but de guider la définition du langage en général, et de l'expression de l'adaptabilité de l'entreprise en particulier.

Ces objectifs sont listés ci-après, et récapitulés de manière structurée dans le Tableau 1. Dans la suite du document, nous ferons indifféremment mention à ces objectifs par leur intitulé ou la référence qui leur est allouée dans le Tableau.

- **Expressivité métier** – Le langage de description d'architecture doit permettre la modélisation de l'architecture cible par et pour les participants à la démarche. En particulier, il doit mettre en œuvre des termes et des concepts qui puissent être expressifs et représentatifs pour des experts opérationnels et non teintés ingénierie.
- **Généricité** – Le langage de description d'architecture ne doit pas être spécifique à un domaine d'application mais doit permettre de décrire des entreprises de domaines métier différents.
- **Maîtrise de la complexité** – Le langage de description d'architecture doit permettre de décrire de manière claire et compréhensible un système complexe, et en particulier un système de systèmes. Ceci doit se traduire par une simplicité du modèle malgré la complexité de l'entreprise modélisée, mais aussi par la simplicité du langage lui-même.
- **Modélisation assistée** – Afin de faciliter la création rapide d'un modèle dans le contexte d'une boucle courte modélisation / exécution, il importe de fournir un maximum de support automatisé à la modélisation. En particulier, nous chercherons autant que possible à automatiser les actions sur le modèle lorsqu'elles relèvent du maintien de la cohérence ou de l'exécutabilité et non d'une décision métier. On veillera ainsi à accorder à l'utilisateur un maximum de liberté quand à la définition et la modification de son architecture sans lui imposer de se préoccuper des problématiques de maintien du modèle.
- **Expressivité architecturale** – Pour permettre l'évaluation de l'adaptabilité de l'entreprise, le langage de description d'architecture doit permettre d'exprimer des points d'articulation dans l'architecture, c'est à dire des points présentent un potentiel de modification.
- **Cohérence** – Pour qu'un modèle fournisse une représentation correcte vis-à-vis du métier, il importe qu'à tout moment il soit maintenu conforme au langage de description d'architecture.
- **Exécutabilité** – Comme évoqué dans la partie précédente, l'approche adoptée relève du domaine de la modélisation exécutable. Un modèle doit ainsi permettre la simulation de processus opérationnels dans leur contexte métier sans traduction vers un modèle de simulation basé sur un formalisme différent.
- **Modification dynamique** – Pour pouvoir représenter de manière dynamique l'adaptabilité de l'entreprise ciblée par l'objectif d'expressivité architecturale, un modèle doit pouvoir être modifié en cours d'exécution.

Intitulé	Désignation	Ref.
Objectifs pour le langage de description d'architecture		
Expressivité métier	Les termes et concepts composant le langage de description d'architecture doivent permettre la modélisation de l'architecture de l'entreprise cible de manière expressive pour des experts opérationnels.	[EL1]
Généricité	Le langage de description d'architecture ne doit pas être spécifique à un domaine d'application.	[EL2]

Maîtrise de la complexité	Le langage de description d'architecture doit permettre de décrire de manière claire et compréhensible un système complexe, et en particulier un système de systèmes.	[EL3]
Modélisation assistée	L'élaboration des modèles doit mettre en œuvre des mécanismes permettant de maintenir automatiquement la cohérence et l'exécutabilité des modèles.	[EL4]
Expressivité architecturale	Le langage de description d'architecture doit permettre la modélisation de points d'articulation dans l'architecture.	[EL5]
Objectifs pour les modèles d'architecture		
Cohérence	Les modèles doivent être conformes au langage de description d'architecture.	[EM1]
Exécutabilité	Les modèles doivent permettre la simulation de processus opérationnels dans leur contexte métier sans traduction vers un modèle de simulation dédié.	[EM2]
Modification dynamique	Les modèles doivent supporter les modifications au cours de leur exécution.	[EM3]

Tableau 1 – Récapitulatif des objectifs de l'approche proposée

1.3.2.2. Application : démarche IDEA

Les travaux présentés dans cette thèse ont été appliqués dans le cadre de la démarche IDEA, développée par Thales Communications & Security. IDEA est une démarche d'architecture dirigée par les modèles, visant à définir, caractériser et évaluer des architectures candidates pour des entreprises répondant à des besoins complexes [39] [40] [14] [15] [43] [38][38].

Le cœur de la démarche est la mise en œuvre d'une approche de prototypage rapide de l'architecture de l'entreprise, sur la base de boucles courtes modélisation / simulation. Les activités de prototypage rapide se font en concertation étroite entre les architectes, les clients et les experts opérationnels.

Comme illustré sur la Figure 4 ci-après, les activités de la démarche s'articulent autour de quatre étapes :

- **Identification** (I) – Identification et analyse du contexte, des problématiques et de l'existant ;
- **Design** (D) – Identification et prototypage d'architectures candidates ;
- **Experimentation** (E) – Evaluation, comparaison et optimisation des architectures candidates par le biais de la simulation ;
- **Assessment** (A) – Sélection d'une architecture parmi les candidates.

Les boucles courtes modélisation / simulation sont en particulier mises en œuvre entre les étapes I et D, et D et E. D'un point de vue temporel, la durée d'une de ces boucles est de l'ordre de quelques heures à une journée.

- Les boucles Identification ⇔ Design (I-D) ont pour objectif la validation de l'architecture. Elles permettent aux différentes parties prenantes (architectes, clients, experts opérationnels) d'élaborer de manière incrémentale une vision commune du besoin, des grandes lignes des variantes d'architectures pouvant y répondre et de leurs critères d'évaluation.

- Les boucles Design ⇔ Experimentation (D-E) ont pour objectif l'évaluation de l'architecture. Elles permettent d'expérimenter les architectures candidates au plus tôt, et de les raffiner graduellement.

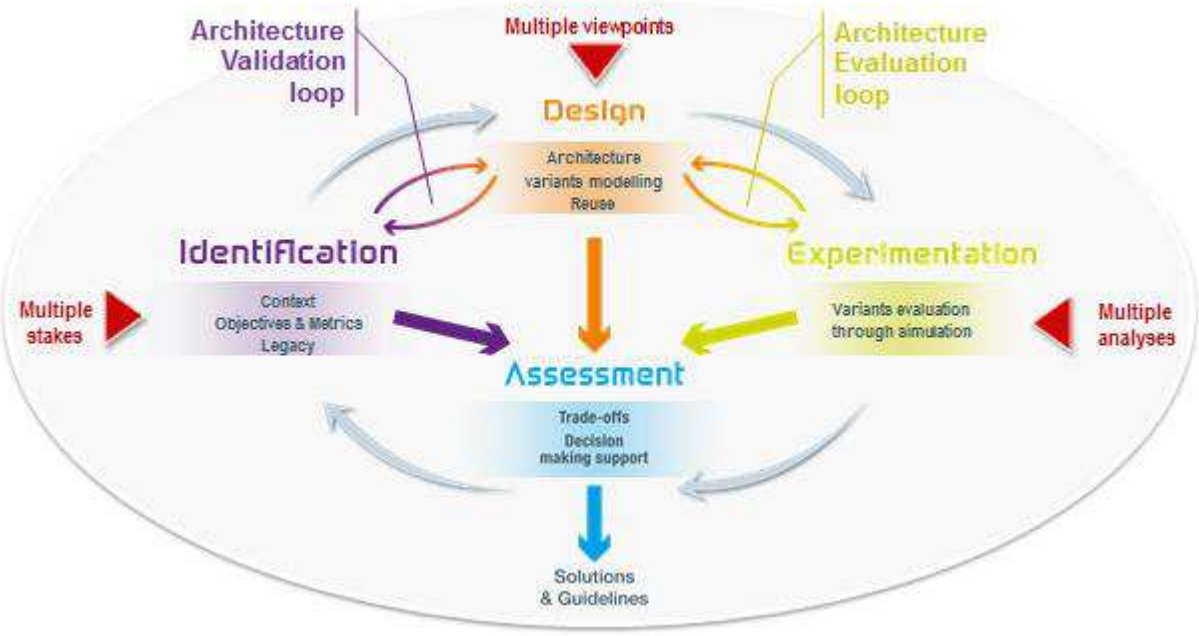


Figure 4 – Démarche d'architecture IDEA [43]

Le support à la démarche IDEA a donné lieu au développement d'une suite d'outils dédiée, basée sur un langage de description d'architecture d'entreprise. Les travaux développés dans le cadre de cette thèse contribuent à la définition de ce langage, en ce qui concerne l'expression de l'adaptabilité.

Des travaux menés sur l'outillage IDEA mais hors du périmètre de cette thèse, en particulier concernant le moteur d'exécution des modèles, seront évoqués lorsque nécessaire à la compréhension d'ensemble mais ne seront pas détaillés.

De part le lien fort existant entre le langage et la démarche du point de vue du projet industriel, le langage de description d'architecture auquel ont été appliqués les travaux de cette thèse sera mentionné comme « langage de description d'architecture IDEA ». Pour abréger, on utilisera également le terme « DSL IDEA ».

1.4. Vue d'ensemble du document

Le Chapitre 2 de ce document présentera les travaux de thèse. Il débutera par une présentation de l'exemple qui sera exploité tout au long de la suite pour illustrer l'exposé, puis par la démarche ayant guidé l'organisation générale du langage. Les deux parties suivantes détailleront les deux contributions majeures de la thèse au projet d'accueil, chacune s'attachant tout particulièrement à présenter les points suivants :

- Le besoin et les constatations ayant motivé ces contributions,
- Les choix faits par rapport à l'état de l'art des travaux de recherche traitant de sujets proches,
- La manière dont les contributions ont été réalisées.

Dans l'ensemble de ce chapitre, nous nous efforcerons de présenter les problématiques et les solutions de manière indépendante de la démarche IDEA et de l'implémentation technique.

Le Chapitre 3 exposera l'implémentation du langage en général et des contributions de thèse en particulier dans la suite d'outils associée à la démarche IDEA. Il s'attachera à en évaluer la mise en œuvre, y compris par le biais de retours d'expérience d'utilisateurs.

Ce chapitre sera constitué de deux parties. La première présentera le retour d'expérience de la mise en œuvre des contributions dans le langage. La deuxième s'appuiera sur le cas concret de l'élaboration d'un modèle dans le cadre d'une affaire pour évaluer les réalisations.

Ce document se terminera par une conclusion et l'exposé des perspectives futures liées aux travaux présentés (Chapitre 4).

Chapitre 2. Expression de l'adaptabilité de l'entreprise dans le langage de description d'architecture

2.1. Préambule

2.1.1. Présentation du cas d'exemple

Les contributions de la thèse seront illustrées sur l'exemple de l'étude des procédures et de l'organisation d'un convoi militaire introduit en 1.1.2. Nous considèrerons un convoi composé de trois camions et quatre unités blindées, chargé de transporter du matériel entre deux points de rendez-vous au cours d'une opération. Une attaque survient au cours du transport, et requiert une réorganisation des éléments du convoi pour faire face à la menace. Le périmètre du modèle comprend l'organisation globale du convoi ainsi que l'organisation interne des unités blindées et des unités de transport, comme synthétisé sur la Figure 5 ci-après.

Le convoi est décomposé en trois groupes fonctionnels, dans lesquels se répartissent les véhicules :

- Le groupe de commandement comprend un chef de convoi et son adjoint (unités blindées). Sa mission est d'assurer le commandement du convoi et les échanges avec les autres unités sur le théâtre.
- Le groupe de transport comprend un chef de groupe et deux transporteurs (camions). Sa mission est le transport des matériels faisant l'objet du convoi.
- Le groupe de protection a pour mission d'assurer la défense du convoi et la surveillance de son environnement. Sa composition dépend de la situation tactique :
    - Dans une situation nominale (déplacement), ce groupe est composé de deux rôles correspondant à la protection de chacun des flancs du convoi. Lorsque le groupe de protection est composé ainsi, le convoi est dit dans sa configuration nominale.
    - Dans le cas d'une situation de combat vient s'y ajouter un rôle de renfort protection. Ce rôle est généralement assuré par l'adjoint du chef de convoi, mais peut également faire intervenir l'appui d'une unité externe au convoi. Lorsque le groupe de protection est composé ainsi, le convoi est dit dans sa configuration de combat (avec ou sans appui d'une unité externe).

Toutes les unités de transport du convoi sont similaires et composées de la façon suivante :

- Un camion, équipé d'un moyen de communications (une radio).
- Un équipage composé de deux membres, un chef d'engin (CC) et un pilote.

Les unités blindées sont composées de la façon suivante :

- Un véhicule blindé, équipé d'une arme principale (un canon), d'une arme secondaire (une mitrailleuse) et d'un moyen de communications (une radio). Le véhicule du chef de convoi est équipé d'une radio spécifique (notée radio CDT), capable de se connecter au réseau de commandement qui lui permet d'échanger avec sa hiérarchie. Les radios des autres véhicules ne leur permettent que d'échanger entre eux et avec le chef de convoi.
- Un équipage composé de deux membres permanents, un chef d'engin / tireur (CTVI) et un pilote, et d'un groupe de combat d'infanterie de 9 fantassins. Le groupe de combat d'infanterie restera embarqué dans le véhicule dans le scénario considéré, aussi son organisation interne ne sera pas prise en compte.

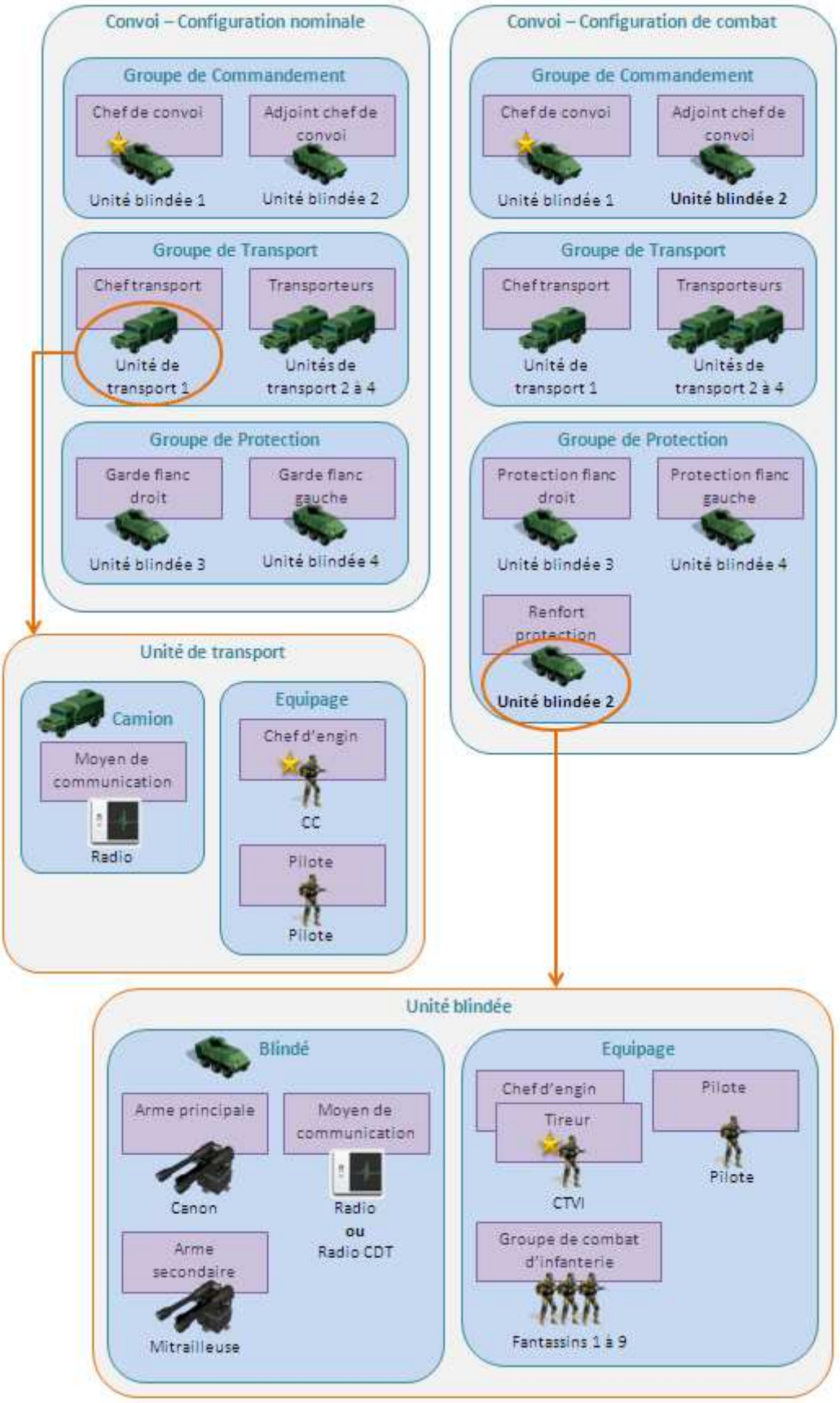


Figure 5 – Synthèse du cas d'exemple

2.1.2. Conventions typographiques

Afin de faciliter la lecture, nous utiliserons pour la suite les conventions typographiques suivantes :

Exemple	Description
Capability	En italique, les noms de concepts, de relations et de propriétés de concepts ou de relations du langage.
CAPABILITY MAP	En majuscules italiques, les noms des vues selon lesquelles sont organisées le métamodèle (la notion de vue sera détaillée plus loin).
ENTERPRISE ORGANIZATION	En majuscules italiques soulignées, les noms des points de vue présentés à l'utilisateur sur le modèle (la notion de point de vue sera détaillée plus loin).
« Convoi »	Entre guillemets et souligné, les noms d'éléments d'un modèle.
`IsCompatibleWith()`	En police `Consolas`, les éléments de code.

Tableau 2– Conventions typographiques

Dans la continuité du chapitre précédent, le gras sera utilisé pour mettre en évidence les points clés.

2.2. Organisation du langage de description d'architecture

2.2.1. Principes de structure du langage de description d'architecture

2.2.1.1. Architecture du langage

Le langage de description d'architecture simulable IDEA est organisé autour de trois composantes distinctes : le métamodèle, des règles dites « de cohérence » et des règles dites « de validation ».

Le métamodèle définit et organise les concepts composant le langage de description d'architecture et les relations entre ces concepts.

Le métamodèle établit la base du langage en termes de sémantique et de grammaire, et dans notre cas doit en particulier assurer l'expressivité métier, la généricité et l'expressivité architecturale du langage ([EL1], [EL2], [EL5]). Par le choix des concepts et de leurs relations, il contribue également à la maîtrise de la complexité ([EL3]).

Les règles de cohérence effectuent des actions sur le modèle afin de lui imposer des règles de structure et de valeur (par exemple d'une cardinalité). Ces règles sont **exécutées automatiquement** sur un événement précis dans la vie du modèle. Elles peuvent **agir sur le modèle** (ex. créer des éléments).

Pour répondre à notre problématique, les règles de cohérence seront en particulier employées pour assurer des actions de maintien automatique de l'exécutabilité du modèle et de conformité au langage ([EL4] pour assurer [EM1] et [EM2]).

Les règles de validation permettent de lister à l'intention de l'utilisateur les incomplétudes ou les erreurs du modèle. Ces règles sont déclenchées de manière interactive, généralement **sur demande de l'utilisateur**. Elles n'effectuent **aucune action sur le modèle**.

Dans le cadre du DSL IDEA, les règles de validation seront principalement utilisées en complément des règles de cohérence lorsqu'il ne sera pas possible, ou pas souhaitable, d'effectuer les actions de maintien de la cohérence et de l'exécutabilité du modèle de manière automatique. De telles situations apparaissent notamment lorsque l'action à mettre en œuvre relève d'un choix d'architecture, devant être pris sur la base de critères propres au domaine métier et non évaluables par un outil.

Les concepts du métamodèle peuvent de plus définir des fonctions servant à la simulation des éléments de modèles qu'ils définissent. Ces fonctions sont mises à disposition du moteur d'exécution, et fournissent par exemple le comportement représenté par un élément de modèle (ex. appel d'un service), ou des règles de parcours du modèle permettant au moteur d'exécution d'avoir accès à toute information pouvant lui être nécessaire. Le moteur d'exécution est pour sa part un composant extérieur au langage.

La Figure 6 ci-dessous présente une vue synthétique de l'architecture du langage et de son interface avec le moteur d'exécution.

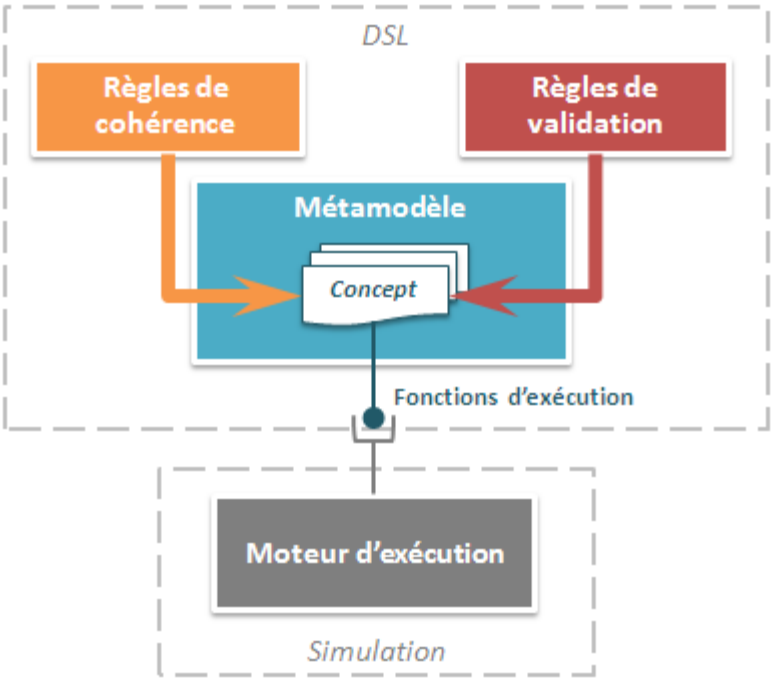


Figure 6 - Architecture du langage et interface avec le moteur d'exécution

Conformité et validité d'un modèle

Il importe de noter que les règles de cohérence et de validation concernent bien la **conformité du modèle par rapport à des principes d'architecture génériques**. Pour permettre l'application du langage à différents domaines métiers, elles ne doivent pas adresser la conformité du modèle par rapport aux principes, à la doctrine ou aux bonnes pratiques du métier dans lequel il s'inscrit.

Par exemple, une règle de validation peut se charger de vérifier que tout service requis par un acteur lui est fourni, mais elle n'évaluera pas la pertinence du choix du fournisseur d'un service par rapport à la doctrine du domaine métier.

Le créateur d'un modèle ne peut pas refuser les actions effectuées par les règles de cohérence, à moins d'annuler l'action ayant mené au déclenchement de ces règles. Le modèle sera donc **toujours conforme aux principes définis par les règles de cohérence**.

En revanche, le créateur d'un modèle peut ne pas tenir compte des messages affichés par les règles de validation, et continuer à travailler et sauvegarder son modèle malgré ces messages : **le modèle peut ainsi être invalide par rapport aux principes définis par les règles de validation**. L'état d'invalidité est un état temporaire, permettant d'apporter une certaine souplesse dans la création du modèle. Ce point sera à nouveau évoqué plus loin dans ce document.

2.2.2. Vue d'ensemble du métamodèle

2.2.2.1. Concepts-clés d'architecture : une brève ontologie du domaine

Comme évoqué dans la partie 1.2, les concepts d'architecture du langage IDEA sont inspirés de ceux manipulés par les Architecture Framework. Pour couvrir le périmètre identifié pour les modèles, le DSL IDEA est articulé autour des concepts définis dans le Tableau 3 ci-après.

Concept métamodèle	Concept opérationnel	Définition
Capability	Capacité	« Aptitude à produire un effet particulier dans des conditions données, par la mise en œuvre d'une combinaison de moyens visant à conduire un ensemble d'activités. » (CJCSM 317001B[11])
Capability Cell	Cellule capacitaire	Un regroupement cohérent de capacités choisies de par leur rapport au même domaine métier.
COI	Communauté d'intérêt	« Un groupement collaboratif d'acteurs qui partagent et échangent des informations pour l'accomplissement de missions ou d'objectifs communs. » (NAF V3 [7])
Enterprise Entity	Entité d'entreprise	Une ressource et un acteur de l'entreprise, logique (agrégat…) ou physique (plateforme, humain…).
Process	Processus	Une activité ou un ensemble d'activités qui mettent en œuvre une capacité en produisant des effets et en transformant des produits en entrée en produits en sortie.
Product	Produit	Un élément, matériel ou informationnel, consommable ou non consommable, créé, utilisé ou échangé dans l'entreprise.
Role	Rôle	Un ensemble de responsabilités attribuées à un acteur ou un groupe d'acteurs au sein d'une collaboration avec d'autres acteurs. A un rôle sont associées des capacités requises pour faire face à ces responsabilités.
Service	Service	« Une fonction, une capacité, ou un comportement effectué par un fournisseur au bénéfice d'un consommateur. » (NAF V3 [7])

Tableau 3 – Concepts-clés du langage de description d'architecture

Le concept de **Communauté d'Intérêt** (CoI – Community of Interest), introduit dans le Tableau précédent, est un concept fondamental vis à vis de l'expression de l'adaptabilité pour les entreprises considérées. En effet, il s'agit d'un concept opérationnel décrivant un **regroupement collaboratif** d'acteurs de l'entreprise, mis en place pour **faire face à une situation particulière**. Une CoI peut être permanente, c-à-d. exister durant l'ensemble du cycle de vie de l'entreprise, ou temporaire, c-à-d être formée pour une durée limitée en réaction à un événement précis. La réponse d'une entreprise à une situation imprévue impliquera généralement la mise en place d'une CoI, dont les membres mettront en œuvre des actions opportunes ou relevant d'une procédure prévue à l'avance pour faire face à cette situation. A noter également que les CoI peuvent être formées de manière transverse à l'organisation nominale des acteurs l'entreprise, c'est à dire faire intervenir des acteurs d'origines hiérarchiques ou organisationnelles diverses.

2.2.2.2. Propriétés domaine et autres caractéristiques communes

Avant de s'intéresser à l'organisation des concepts-clés listés dans le Tableau 3 dans un métamodèle de description d'architecture d'entreprise, il convient de poser les bases de ce

[11] *Chairman of the Joint Chiefs of Staff Manual*, 2007 – Document de référence terminologique en ce qui concerne les missions interarmées américaines.

métamodèle. Indépendamment du concept dont ils sont issus, on constate en effet que tous les éléments d'un modèle doivent être dotés des caractéristiques suivantes :

- Trivialement, ils doivent pouvoir porter un nom.
- Ils requièrent d'être identifiés de manière unique dans un modèle. A priori, le nom d'un élément n'est pas suffisant : par exemple, les capacités sont couramment mentionnées en utilisant un verbe, et dans ce cas il n'est pas illogique d'attribuer le même nom à l'élément *Capability* et à l'élément *Process* qui décrit sa mise en œuvre (« Commander », « Observer »…). Les éléments doivent donc être dotés en plus de leur nom d'un identifiant unique.
- Ils sont susceptibles d'être dotés de propriétés ayant trait à la modélisation et/ou à la documentation du modèle : par exemple, la date de création d'un élément, ou un commentaire de l'utilisateur à son sujet.
  Dans la suite de cette thèse, on utilisera le terme **métadonnées** pour référer à ce type de propriétés.
- Ils sont susceptibles d'être dotés de propriétés représentant les propriétés métier du constituant modélisé : par exemple, la localisation géographique d'un bâtiment. Le type de ces propriétés relève du modèle de données correspondant au domaine de l'entreprise (ex. localisation d'un bâtiment par son adresse postale ou sous forme de coordonnées GPS), et est à définir dans le modèle.
  Dans la suite de cette thèse, on utilisera le terme **propriétés domaine** pour référer à ce type de propriétés.

Les choix de modélisation des métadonnées et des propriétés domaine dans le DSL ne relevant pas du cadre de cette thèse, ils ne seront pas détaillés dans ce document. Ils sont cependant présentés à titre d'information en Annexe 1.

2.2.2.3. Organisation du métamodèle et expressivité architecturale

Afin d'optimiser la maîtrise de la complexité du langage lui-même et faciliter à la fois sa prise en main par les utilisateurs et sa maintenance par les développeurs du langage, il importe d'organiser le métamodèle de manière claire et structurée.

Dans une optique d'expressivité architecturale [EL2], nous avons choisi de calquer l'organisation du métamodèle sur celle de l'architecture d'entreprise. Contrairement au découpage adopté dans les Architecture Frameworks, cette approche de l'organisation du métamodèle ne cherche pas à décrire l'architecture selon les perspectives des différents intervenants. Elle se base au contraire sur la distinction entre les différents aspects internes de l'entreprise, comme les différents diagrammes UML. Le principal intérêt de cette approche est de **mettre en exergue dans le langage même les points d'articulation** majeurs de l'entreprise, afin de faciliter leur identification par l'utilisateur et leur exploitation pour l'expression de l'adaptabilité. Par exemple, ils doivent fournir un support à la comparaison de différentes organisations d'acteurs pour obtenir les mêmes capacités, ou à l'étude de la possibilité d'exploiter les mêmes acteurs dans le cadre de différents schémas de collaboration.

Afin d'homogénéiser le langage dans une optique d'exécution et de minimiser sa complexité pour l'utilisateur ([EL3]), le DSL IDEA utilise les mêmes concepts aux différents niveaux de granularité de l'architecture. Les Architecture Frameworks introduisent les vues opérationnelles et systèmes où l'on retrouve parfois des concepts et des formalismes semblables. Ce n'est pas le cas dans le DSL IDEA, dans lequel le même jeu de concept s'applique aux différents niveaux de granularité. Par exemple, le concept *Capability* du DSL IDEA pourra aussi bien être utilisé pour décrire la capacité collaborative

d'élaboration d'image perçue (niveau NAF opérationnel), que pour la capacité de mobilité d'un véhicule ou la capacité d'observation de nuit d'une caméra (niveau NAF système).

Suivant ces principes, le DSL IDEA est organisé autour de quatre vues, organisant chacune un ensemble de concepts et de relations permettant de répondre à une problématique élémentaire de description de l'architecture (qui ? quoi ? comment ? avec quoi ?). Ces quatre vues et les principaux concepts et relations qui les composent sont résumés dans la Figure 7.

Concrètement, ces quatre vues se traduisent dans le code généré à partir du métamodèle par quatre espaces de nommage regroupant les classes décrivant les concepts liés à chaque vue et les relations associées. Dans l'éditeur de modèle, chacune des vues du métamodèle correspond à un ensemble de diagrammes distincts qui seront présentés dans la partie 3.1.1.

- **« Qui ? »** – La vue **ENTREPRISE ORGANIZATION** décrit l'organisation des acteurs de l'entreprise (entités d'entreprise).
  Ses principaux concepts et relations seront exposés dans la partie 2.2.2.4.
- **« Quoi ? »** – La vue **CAPABILITY MAP** décrit les capacités de l'entreprise, ainsi que les processus mis en œuvre pour les atteindre ainsi que les services qui les exposent.
  Ses principaux concepts et relations seront exposés dans la partie 2.2.2.5.
- **« Comment ? »** – La vue **ROLES & COLLABORATIONS** décrit les schémas de collaborations établis entre les rôles identifiés dans l'entreprise.
  Ses principaux concepts et relations seront exposés dans la partie 2.2.2.6.
- **« Avec quoi ? »** – La vue **COMMONS** décrit les produits (physiques ou informationnels) exploités par l'entreprise et le modèle de données utilisé pour les propriétés domaine.
  Comme évoqué précédemment, les propriétés domaine ne relèvent pas du cadre de cette thèse, mais la vue Commons sera présentée à titre d'information en Annexe 1.

Le métamodèle prend en compte trois points d'articulations majeurs de l'architecture, représentés par trois relations établies entre des concepts appartenant à des vues différentes :

- La relation ***Requires*** entre les concepts *Role* et *CapabilityCell* traduit les exigences en termes de capacités attendues par le rôle vis-à-vis des acteurs qui pourraient le jouer.
- La relation ***Has*** entre les concepts *EnterpriseEntity* et *CapabilityCell* traduit les capacités propres à un acteur. Ces capacités peuvent ou non être exploitées par l'acteur pour jouer un rôle.
- La relation ***Plays*** entre les concepts *EnterpriseEntity* et *Role* représente la prise en charge par un acteur des responsabilités définies par un rôle au sein d'une collaboration.

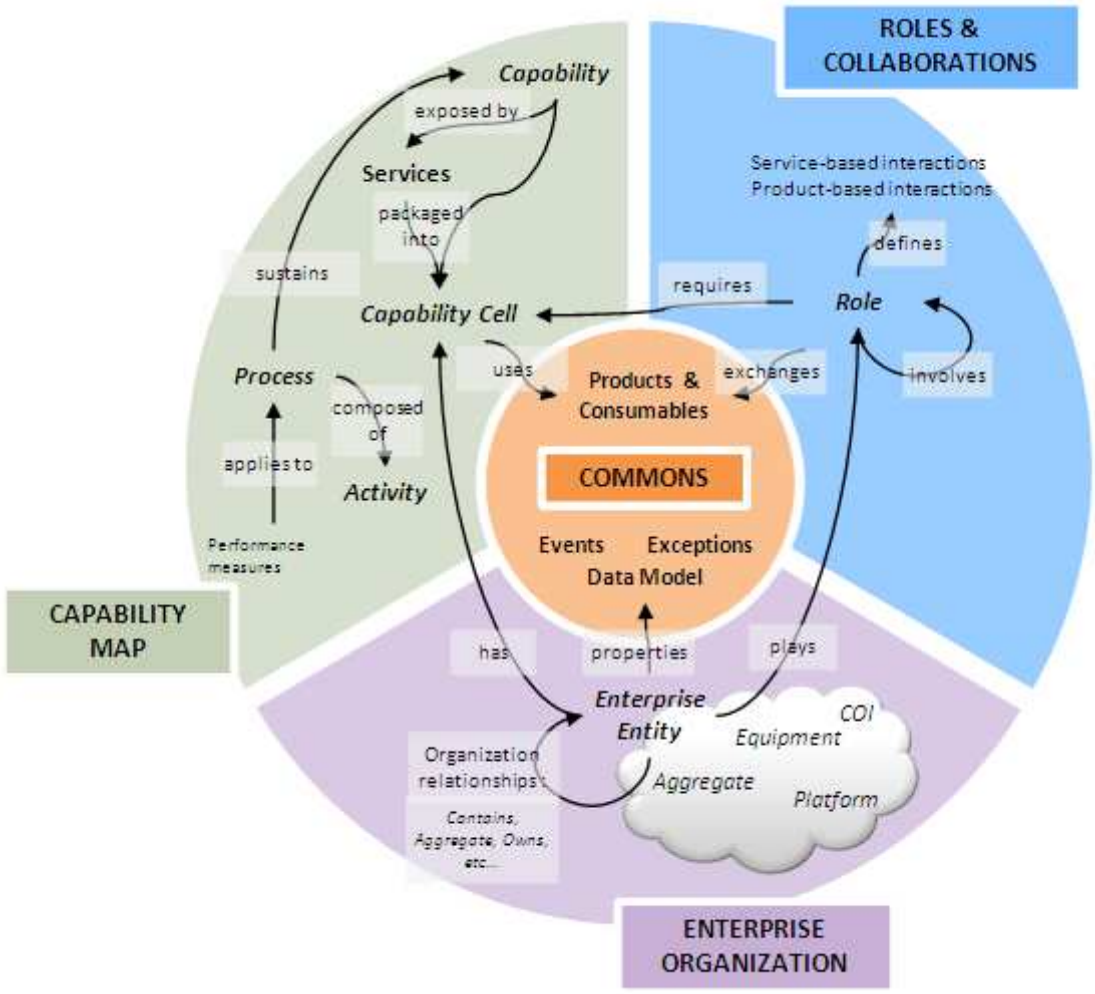


Figure 7 – Principe d'organisation du métamodèle IDEA [43]

2.2.2.4. Vue ENTERPRISE ORGANIZATION

Les principaux concepts et relations de la vue ENTERPRISE ORGANIZATION sont présentés dans la Figure 8, les concepts-clés définis en 2.2.2.1 étant illustrés en couleur.

Dans la suite de cette thèse, on s'intéressera en particulier aux relations de décomposition des entités :

- Relations de **décomposition hiérarchique** – La relation *Contains*, qui traduit l'embarquement d'une entité sur une entité physique (*PhysicalEnterpriseEntity*), et la relation *Aggregates*, qui traduit l'appartenance d'une entité à un groupe logique (*Aggregate*). Ces relations expriment le fait que l'entité cible fait partie intégrante de l'entité source. Par exemple dans le modèle de convoi, la relation entre un véhicule et sa caméra embarquée est modélisée par une relation *Contains.* La relation entre une unité blindée et le convoi est modélisé par une relation *Aggregates*.

  En général, une entité peut être cible de plusieurs relations hiérarchiques. Ce point sera exposé et discuté en détail en 2.3, en particulier par rapport à l'exécution du modèle.
- Relations de **décomposition transverse** – La relation *Hosts*, qui traduit l'hébergement d'une entité dans une entité physique (*PhysicalEnterpriseEntity*), et la relation *Gathers*, qui traduit la

participation d'une entité à une Communauté d'Intérêt (*COI*). Ces relations sont établies de manière orthogonale aux relations de décomposition hiérarchique.

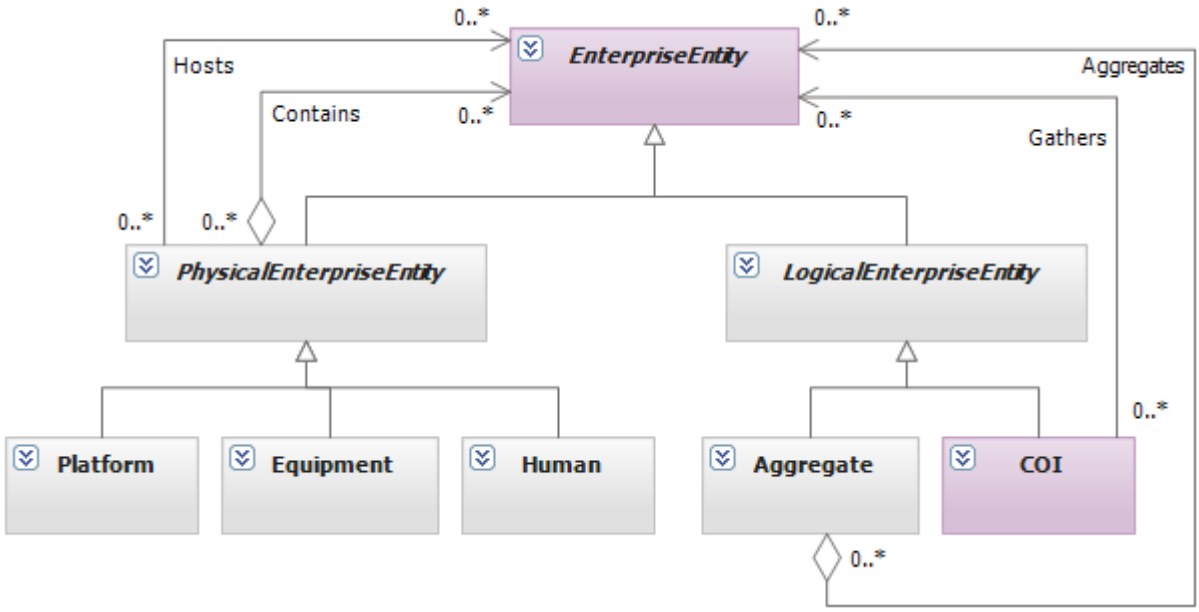


Figure 8 - DSL IDEA : principaux concepts et relations de la vue ENTERPRISE ORGANIZATION

Par exemple, une unité de transport du convoi peut se modéliser de la manière suivante, illustrée sur la Figure 9 :

- Un *Aggregate* nommé « Unité de transport », lié par des relations *Aggregates* à :
  - Une *Platform* « Camion », liée par une relations *Contains* à un *Equipment* « Radio »,
  - Un *Aggregate* « Equipage », lié par des relations *Aggregates* à deux *Human* « CC » et « Pilote ».
- Pour exprimer le fait que l'équipage est embarqué dans le camion, la *Platform* « Camion » est de plus liée par une relation *Hosts* à l'*Aggregate* « Equipage ».

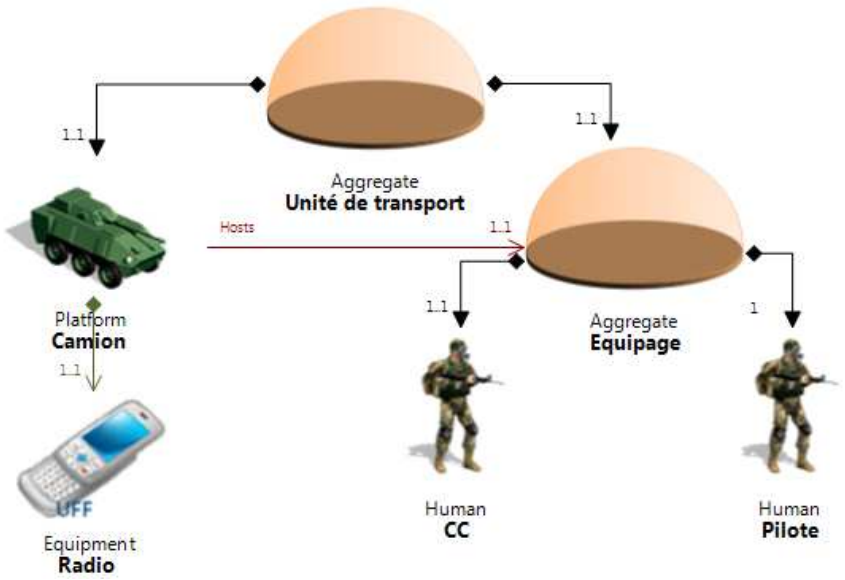


Figure 9 – Application des concepts de la vue ENTERPRISE ORGANIZATION au modèle du convoi

2.2.2.5. Vue CAPABILITY MAP

Les principaux concepts et relations de la vue CAPABILITY MAP sont présentés dans la Figure 10, les concepts-clés définis en 2.2.2.1 étant illustrés en couleur.

On notera en particulier que la notion de service considérée ici englobe aussi bien les services opérationnels (ex. un service d'appui feu) que les services techniques. Le DSL IDEA est inspiré du paradigme d'Architecture Orientée Service [41] [41] pour les représenter, mais de manière simplifiée pour ne pas complexifier le langage pour les utilisateurs. L'interface d'un service est ainsi constituée de ses opérations, représentées par des capacités. On peut considérer l'implémentation d'un service comme l'ensemble des processus de ses opérations.

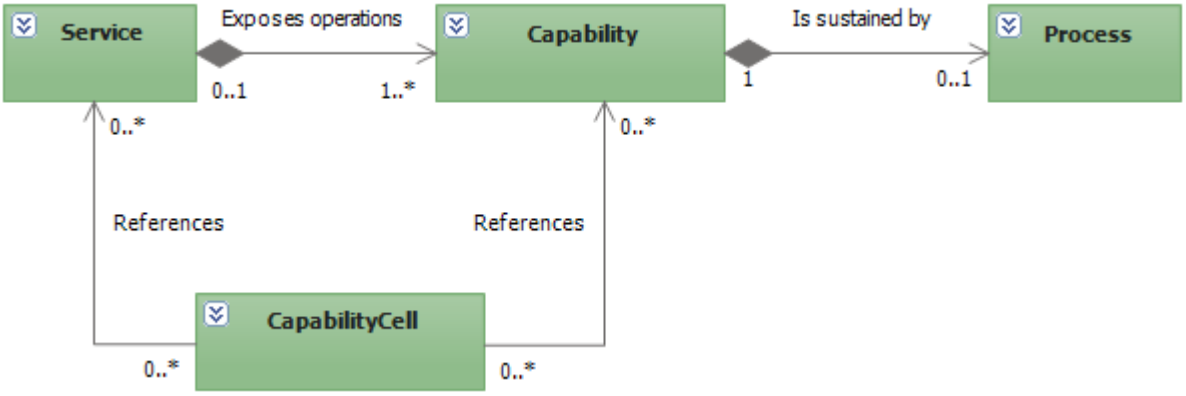


Figure 10 - DSL IDEA : principaux concepts et relations de la vue CAPABILITY MAP

Par exemple, on peut considérer que le convoi dans son ensemble est doté de trois capacités, « Transporter du matériel », « Commander le convoi » et « Assurer la protection du convoi ». Si cette dernière capacité a une vocation purement interne au convoi, les deux premières doivent être exposées à l'extérieur pour pouvoir interagir : le transport de matériel se fait au profit d'un demandeur, et les interactions avec ce demandeur se font par le biais du commandement du convoi (ex. modification d'itinéraire…). Le modèle comporte ainsi un *Service* « Transport par convoi », référençant en tant qu'opérations (relations *ExposesOperations*) les capacités « Transporter du matériel » et « Commander le convoi ».

2.2.2.6. Vue ROLES & COLLABORATIONS

Les principaux concepts et relations de la vue ROLES & COLLABORATIONS sont présentés dans la Figure 11, les concepts-clés définis en 2.2.2.1 étant illustrés en couleur. De manière similaire à la décomposition hiérarchique des entités, les rôles se décomposent en d'autres rôles de granularité plus fine, par le biais de la relation *Involves*.

La décomposition des rôles fait intervenir la notion de collaboration. Dans ce cadre, nous utiliserons le terme collaboration pour désigner un ensemble de rôles contenus dans un même rôle de granularité supérieure selon des relations Involves. Par abus de langage, un rôle peut alors être considéré comme une collaboration entre les rôles qu'il contient.

Le concept de *Role* est d'un intérêt fondamental en termes d'articulation de l'entreprise. En effet, il établit par sa structure même la **flexibilité entre les organisations et les processus de l'entreprise** : ce sont sur ce concept et les notions associées que portent l'essentiel des contributions de cette thèse.

Une collaboration peut faire intervenir entre les rôles qu'elle concerne des interactions de service (*ServiceInteraction*), des échanges de produits (*ProductInteraction*) et d'évènements (*EventInteraction*, similaire à *ProductInteraction* et non représenté sur la Figure 11).

La manière dont un rôle peut interagir est définie *via* le concept *Endpoint*, représentant un service que le rôle peut fournir (*ProvidedServiceEndpoint*), un service qu'il consomme (*ConsumedServiceEndpoint*) ou un produit ou événement opérationnel qu'il peut échanger (*SentProductEndpoint* et *ReceivedProductEndpoint*, et *SentEventEndpoint* et *ReceivedEventEndpoint* non représentés sur la Figure 11 pour des raisons de lisibilité).

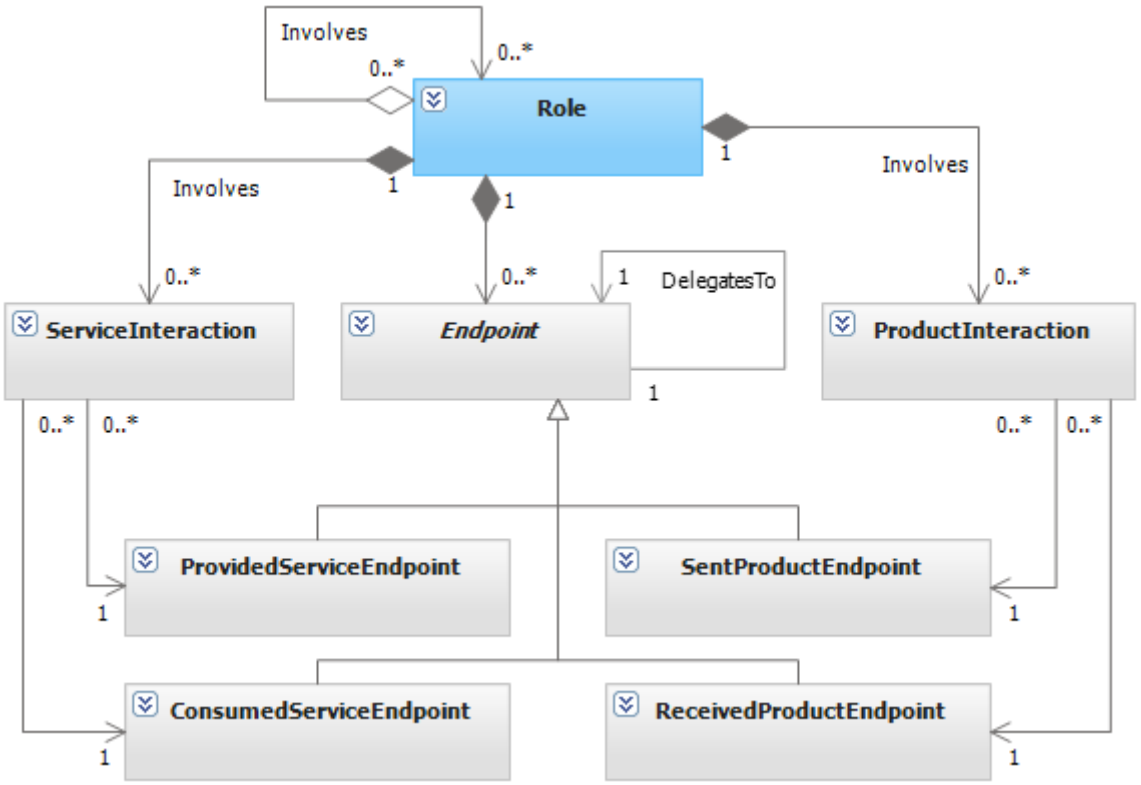


Figure 11 - DSL IDEA : principaux concepts et relations de la vue ROLES & COLLABORATIONS

Par exemple, l'organisation du convoi en groupes dans sa configuration nominale peut être modélisée sous forme de rôles, comme illustré sur la Figure 12 sous forme d'arborescence :

- Un *Role* « Convoi », décomposé *via* des relations *Involves* en trois autres *Roles*, « Groupe de commandement », « Groupe de transport » et « Groupe de protection »,
- Le « Groupe de commandement » est lui-même décomposé en deux *Roles*, « Chef de convoi » et « Adjoint chef de convoi »,
- Le « Groupe de transport » est décomposé en plusieurs *Roles* pour le « Chef transport » et les « Transporteurs »[12],
- Le « Groupe de protection » est décomposé en deux *Roles* « Garde flanc droit » et « Garde flanc gauche ».

[12] La manière de modéliser le fait que le groupe compte entre 1 et 4 transporteurs sera présentée ultérieurement.

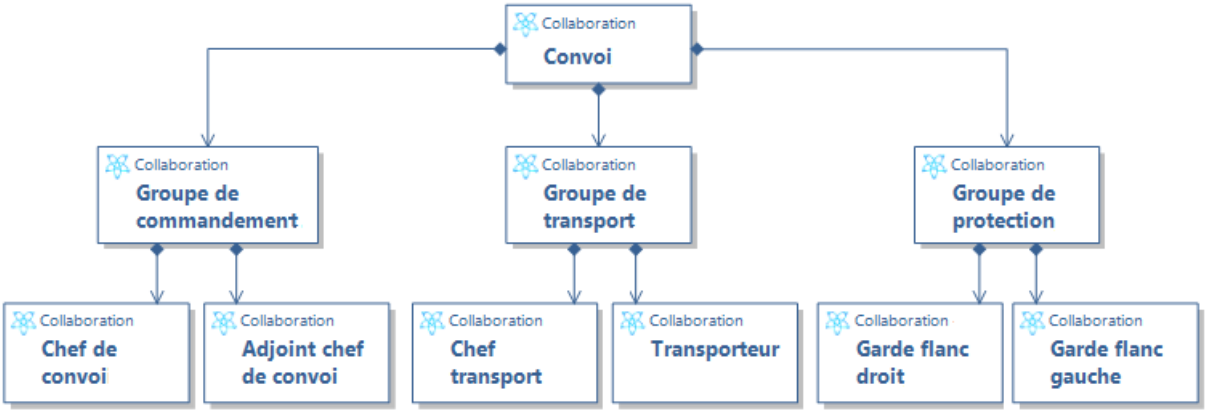


Figure 12 – Application des concepts de la vue "Roles & Collaboration" sur le modèle du convoi (1)

Le « Groupe de Commandement » envoie des ordres aux deux autres groupes, qui lui renvoient des compte-rendus. Cet échange est modélisé par des *ProductInteraction*, établie au niveau de la collaboration « Convoi » entre des *Endpoints* situés sur les interfaces des rôles des différents groupes, comme illustré sur la Figure 13.

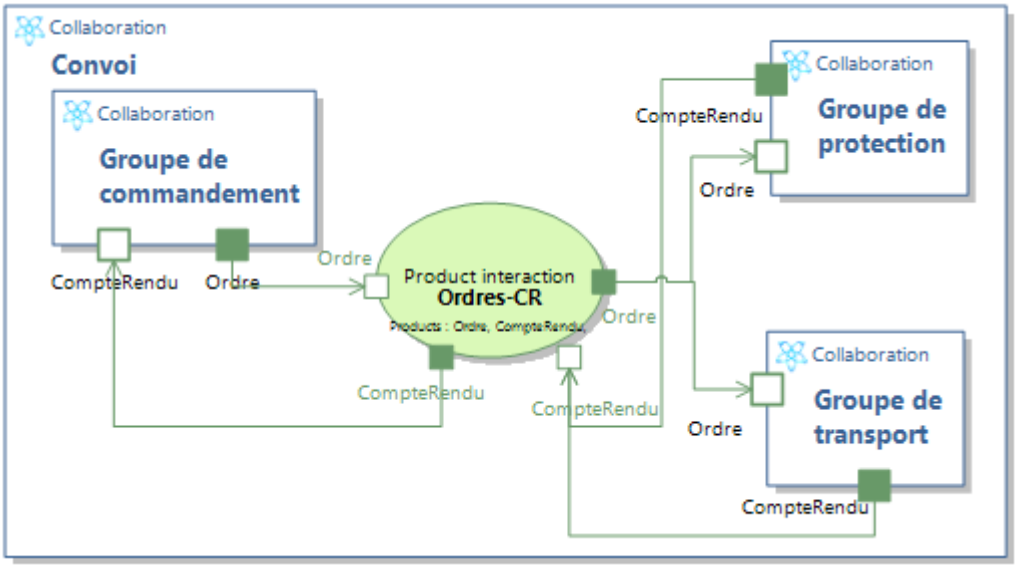


Figure 13 – Application des concepts de la vue "Roles & Collaboration" sur le modèle de convoi (2)

Au sein du « Groupe de commandement », c'est en réalité le rôle du « Chef de convoi » qui envoie l'ordre qui doit transiter jusqu'au « Groupe de transport » et au « Groupe de protection » et qui réceptionne les comptes-rendus. C'est pour exprimer cette précision qu'a été introduite la relation ***DelegatesTo***, présente au niveau du concept *Endpoint* sur la Figure 11. La Figure 14 illustre l'utilisation d'une telle relation, représentée en pointillés, sur l'exemple de l'envoi de l'ordre et la réception des comptes-rendus par le « Chef de convoi » au sein du « Groupe de commandement ».

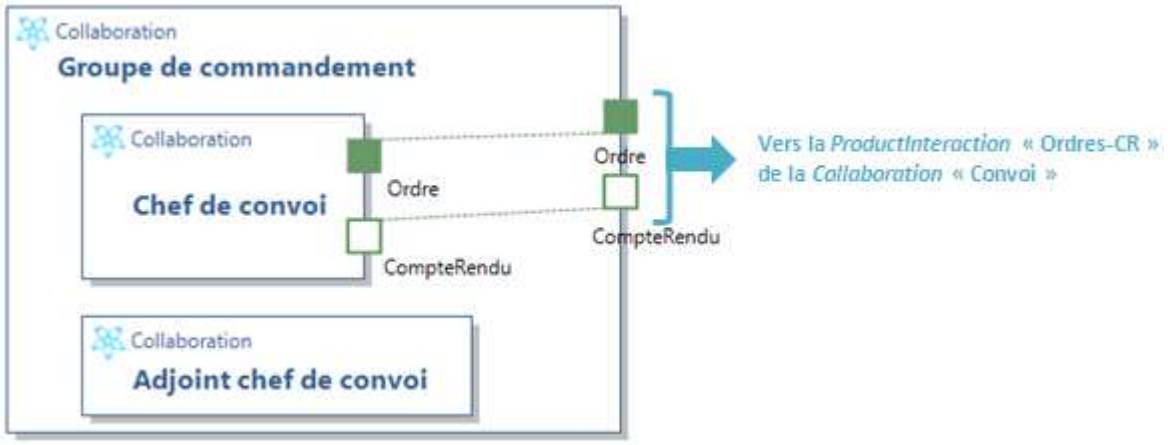


Figure 14 – Application des concepts de la vue "Roles & Collaboration" sur le modèle de convoi (3)

2.2.2.7. Cas particulier du concept *COI*

Comme évoqué précédemment, le concept opérationnel de COI est central dans l'expression de l'adaptabilité de l'entreprise, dans la mesure où il représente un regroupement d'acteurs dédié à une situation particulière. Dans le DSL IDEA, il est représenté par le concept *COI*, appartenant à la vue ENTERPRISE ORGANIZATION. La définition de ce concept fait intervenir une relation particulière vers le concept *Collaboration*, nommée *Implement* et illustrée par la Figure 15. Cette relation traduit le lien vers **les interactions et les procédures spécifiques qui doivent être mises en œuvre** par les membres de la COI pour réaliser leur mission.

Cette relation représente un **potentiel** pour la relation *Plays* : dans le cas d'une COI temporaire, la relation *Implements* ne traduira en effet que des interactions et des procédures attendues aussi longtemps que la COI ne sera pas mise en place. Lorsque ce sera le cas, la relation *Implements* équivaudra à un *Plays*[13].

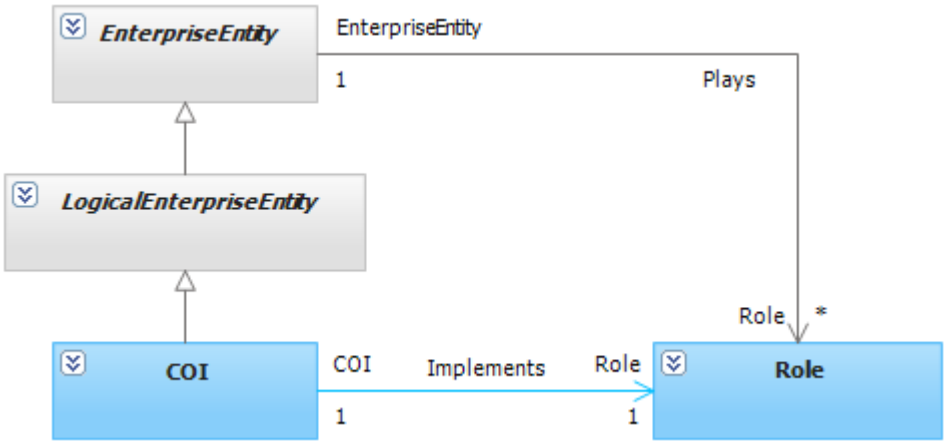


Figure 15 – DSL IDEA : focus sur les concepts de *COI* et de *Role* (*Collaboration*)

[13] Bien qu'étant établie entre des concepts appartenant à une vue différente, cette relation n'a par conséquent pas été représentée sur la Figure 7 et identifiée comme point d'articulation majeur.

2.2.3. Principe général d'exécution d'un modèle

Comme évoqué en 1.3.1.2, l'objectif en termes d'exécution de modèle est de simuler l'architecture dans son ensemble sans transformation de modèle. Le moteur de simulation sortant du cadre de cette thèse, l'exécution des modèles ne sera pas abordée en détail.

Le principe choisi pour ce faire est la simulation de **processus** (concept *Process*) en prenant en compte leur **contexte métier**, c'est-à-dire quel membre de l'entreprise les déroule (concept *EnterpriseEntity*) et selon quel schéma d'interaction (concept *Collaboration*).

Par exemple, considérons deux processus consistant respectivement à « Commander » (envoyer un ordre et recevoir un compte-rendu) et à « Exécuter » (recevoir un ordre et envoyer un compte-rendu). Dans le modèle de convoi, comme illustré sur la Figure 16[14], les rôles du « Chef de convoi » et du « Chef transport » peuvent dérouler le processus « Commander », et les rôles de « Chef transport » et de « Transporteur » peuvent dérouler le processus « Exécuter ». Les échanges d'ordres et de compte-rendus entre ces trois rôles ne se font au sein du convoi que selon les flux représentés sur la Figure 16.

Lorsqu'un processus « Commander » est exécuté, il est indispensable de prendre en compte les échanges définis entre les rôles dans la collaboration « Convoi » pour pouvoir déterminer le destinataire de l'ordre envoyé : si le processus est exécuté dans le contexte du rôle « Chef de convoi », le destinataire sera le « Chef transport », et s'il est exécuté dans le contexte du « Chef transport », le destinataire est le « Transporteur ».

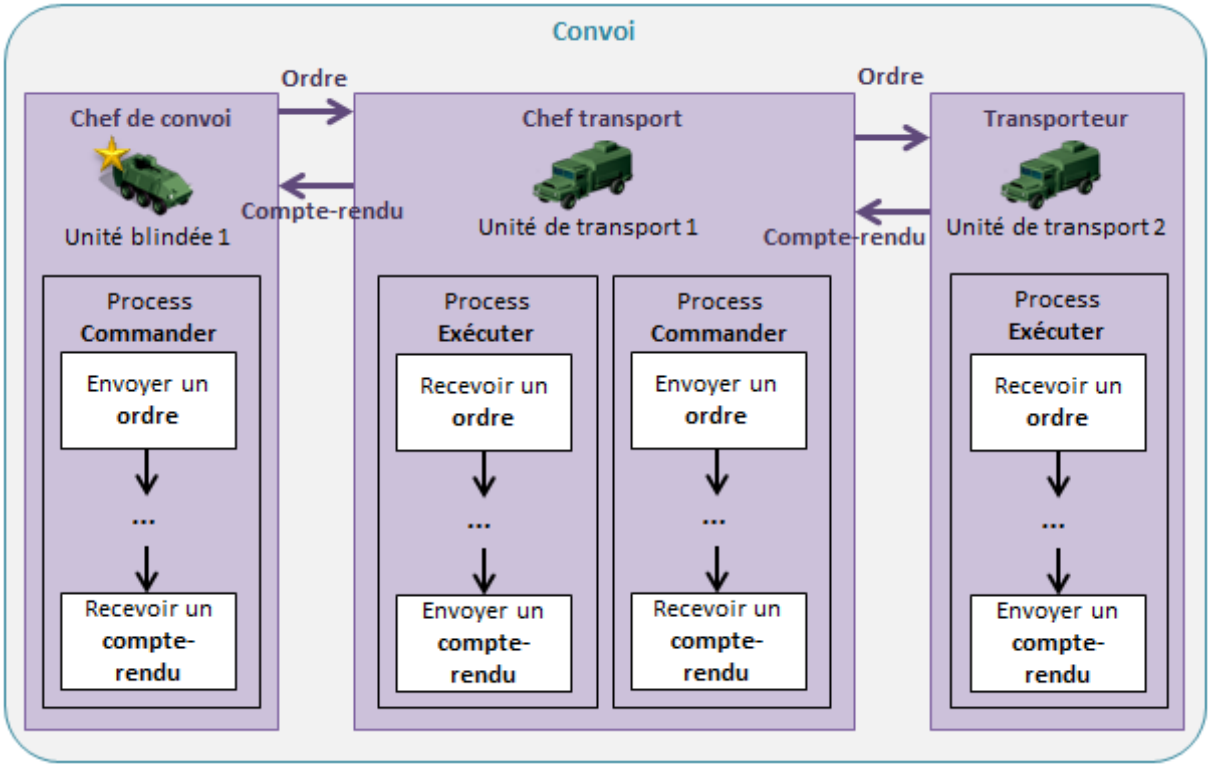


Figure 16 - Exemple d'interactions sur le modèle de convoi

[14] Pour éviter de complexifier inutilement le schéma, la Figure 16 ne comprend pas les *Roles* « Groupe de commandement » et « Groupe de transport ». Ceux-ci sont inclus dans le « Convoi », et contiennent respectivement le rôle « Chef de convoi » et les rôles « Chef transport » et « Transporteur » (cf. Figure 12). Ils n'ont aucune influence sur les propos tenus ici.

2.3. Types et instances domaine

2.3.1. Analyse du besoin

2.3.1.1. Notions de types & instances domaine

En phase de définition de l'architecture[15], l'utilisateur construit le modèle ou une partie du modèle, au niveau de granularité de son choix. A ce stade, le besoin de modélisation correspond principalement à la définition des constituants d'architecture et de la manière dont ils s'articulent. Dans le cas où plusieurs constituants similaires sont à prendre en compte à des endroits différents de l'organisation, il apparaît important de les matérialiser dans le modèle par un élément unique. La réutilisation de cet élément unique permet de capitaliser la définition des constituants identiques et ainsi d'éviter toute erreur de mise en cohérence en cas de modification des caractéristiques communes. Par exemple, les quatre unités blindées du convoi sont strictement identiques en termes d'organisation et de capacités. Dans le modèle, il est plus intéressant de disposer d'une définition unique d'une telle unité blindée plutôt que de définir les quatre unités de manière redondante.

En phase d'exécution du modèle[16], l'utilisateur exploite la simulation du comportement de l'entreprise. A ce stade, il a besoin d'individualiser les constituants de l'architecture déployée pour représenter les différents éléments du modèle sur un théâtre d'exécution virtuel. Ce n'est qu'ainsi que les éléments, pions individuels de la simulation, pourront avoir des valeurs de propriétés et des comportements spécifiques.

Sur l'exemple des processus du convoi introduit en 2.2.3, il est indispensable lors de la simulation d'un processus « Commander » de savoir s'il s'agit d'un processus du rôle « Chef de convoi » ou du rôle « Chef transport » pour s'assurer que l'ordre envoyé arrive au destinataire adéquat. Il importe donc de pouvoir les distinguer facilement et rapidement : c'est chose aisée si le processus « Commander » du « Chef de convoi » et celui du « Chef transport » sont deux éléments de modèle différents. Dans le cas des processus, ce besoin est par ailleurs d'autant plus important que deux rôles peuvent chercher à exécuter un même processus en même temps au cours de la simulation, sans en être au même stade : par exemple, le « Chef de convoi » et le « Chef transport » peuvent être tous deux en train de dérouler un processus « Commander », l'un en étant à l'envoi d'un ordre, l'autre à la réception d'un compte-rendu.

Un modèle simulable au sens de la définition introduite en 1.3.1.2 est supposé pouvoir répondre à ces deux besoins, et donc représenter à la fois :

- Des **types de constituants d'architecture**, c.-à-d. des éléments capitalisés représentant un ensemble de constituants identiques, que l'on mentionnera sous la dénomination de **types domaine**,
- Et des **constituants d'architecture déployés**, c.-à-d. individualisés pour la simulation, les **instances domaine**.

Un type d'élément représente alors la description des caractéristiques communes à un ensemble d'instances de même nature.

[15] Par exemple, lors de l'étape D de la démarche IDEA introduite en 1.3.2.2
[16] Par exemple, lors de boucles I-D ou D-E de la démarche IDEA

2.3.1.2. Types & instances domaines pour le DSL IDEA

Pour illustrer plus précisément les notions de type et d'instance domaine, et pour identifier le périmètre qu'elles recouvrent dans le métamodèle, nous allons chercher à les rapprocher de la réalité des entreprises considérées :

- Dans le cadre des *EnterpriseEntity*, le principe de type domaine est similaire aux classes de navires. Une classe de navires définit une série de bâtiments construits selon les mêmes plans, partageant les mêmes caractéristiques et pouvant effectuer les mêmes activités. Par exemple, les remorqueurs de haute mer Abeille Bourbon et Abeille Liberté ont les mêmes propriétés techniques (longueur, tirant d'eau …), les mêmes équipements (propulseurs, radars, transmissions…) et ont la capacité à remplir les mêmes missions (intervention, d'assistance et de sauvetage de navires en détresse).

  Par analogie, dans un modèle une *EnterpriseEntity* de niveau type domaine définit une série d'instances domaine partageant les caractéristiques suivantes en termes de modélisation :
  - Propriétés domaine, tant en termes de définition que de valeurs, par exemple pour définir la longueur ou le tirant d'eau des navires d'une même classe,
  - Décomposition en sous-entités selon les relations de décomposition, par exemple pour définir les équipements des navires d'une même classe,
  - *CapabilityCells* (selon les relations *EnterpriseEntity Has CapabilityCells*), par exemple pour définir la capacité des navires d'une même classe au sauvetage en haute mer.
- Dans le cadre des *Roles,* le principe de type domaine est similaire à celui des procédures, par exemple de la doctrine militaire. Ces procédures visent décrire la manière de répondre à une certaine situation, par le biais de la mise en place d'activités dans un ordre clairement défini, faisant intervenir des rôles identifiés se coordonnant par des échanges d'informations selon un schéma précis. Lorsque la situation apparait dans la réalité, les acteurs qui lui font face reproduisent alors le modèle de collaboration et suivent le processus défini par la doctrine.

  Par analogie, dans un modèle une collaboration de niveau type domaine définit la procédure à mettre en place dans une certaine situation, selon les caractéristiques suivantes en termes de modélisation :
  - Propriétés domaine, par exemple pour définir le délai minimum sous lequel la collaboration doit être mise en place,
  - Décomposition en rôles et en interactions selon les relations *Involves*, pour identifier les rôles et la manière dont ils se coordonnent dans le cadre de la procédure.

La notion de type domaine peut ainsi englober un ensemble d'éléments (ex. une collaboration et les rôles qui la composent). Les instances domaine correspondant à ce type seront représentées par un ensemble d'éléments liés entre eux de manière identique.

Dans la suite du document, nous utiliserons la terminologie suivante :

- **Type domaine** : un élément ou un ensemble d'éléments de modèle capitalisant une définition commune à un ensemble de constituants d'architecture identiques ;
- **Instance domaine** : un élément ou un ensemble d'éléments de modèle représentant un constituant d'architecture déployé, individualisé pour la simulation, dont les caractéristiques sont conformes à un type domaine ;

- **Instanciation** : la génération des instances domaine à partir des types domaine. Nous utiliserons le terme **déploiement** pour désigner l'ensemble des instances domaines résultant d'une instanciation.

La distinction établie entre les types et les instances domaines permet l'**exploitation des points d'articulation** du métamodèle. En effet, les types domaines permettent de représenter une articulation nominale ou prévue, que les instances domaines permettront de modifier afin d'expérimenter à volonté différentes structures. Par exemple, la modification de l'*EnterpriseEntity* de niveau instance domaine à l'origine d'une relation *Plays* permet d'évaluer l'impact de la prise en charge d'un rôle donné par un acteur ou un autre dans le cadre d'un mode dégradé, sans pour autant perdre l'information liée au mode nominal (capitalisée au niveau des types domaine).

Cette distinction permet également de représenter de manière fidèle la **dynamicité** organisationnelle apportée par le concept de *COI*. La modélisation d'une *COI* de niveau type domaine et de la *Collaboration* qui lui est liée par une relation *Implements* traduit en effet dans l'entreprise une option prévue pour faire face à une situation particulière. La génération d'une instance de cette *COI* équivaut opérationnellement à sa mise en place, et doit pouvoir intervenir à tout moment du cycle de vie du modèle, en particulier **en cours de simulation**.

Du point de vue du langage de description d'architecture, la mise en œuvre du principe types / instances domaine relève de la problématique de la modélisation multiniveau. Différentes méthodes d'implémentation de ces notions dans le langage de description d'architecture seront présentées dans la partie 2.3.2. Les choix adoptés dans le cadre d'IDEA seront par la suite illustrés sur l'exemple des concepts *EnterpriseEntity* et *Collaboration*.

2.3.1.3. Automatisation de l'instanciation

En support à la modélisation, il importe pour une approche de prototypage rapide d'automatiser autant que possible les actions sur le modèle lorsqu'elles relèvent du maintien de la cohérence et non d'une décision métier. Dans cette optique, nous avons besoin de définir et d'exploiter des algorithmes visant à **générer automatiquement un déploiement** à partir d'un ensemble de types domaine.

A un ensemble donné de types domaine peuvent correspondre plusieurs déploiement différents, en particulier si des cardinalités sont introduites dans l'organisation des types. L'instanciation automatique devra donc adopter des règles d'instanciation par défaut pour résoudre les cas pouvant donner lieu à plusieurs déploiements différents.

Les règles d'instanciation par défaut seront définies de manière déterministe, afin que les résultats de deux instanciations automatiques d'un même patron de types domaine soient similaires. Cette approche, par opposition à des règles basée sur un comportement aléatoire, permet à l'utilisateur d'avoir une idée *a priori* du résultat d'une instanciation par défaut et de conserver un maximum de maîtrise sur l'architecture modélisée.

Par exemple, le convoi peut contenir entre deux et quatre unités de transport. Il y a donc trois déploiements possibles, contenant respectivement deux, trois et quatre instances domaine de l'unité de transport. Par défaut, l'instanciation automatique tiendra compte de la valeur minimale de la

cardinalité et produira ainsi un déploiement contenant deux instances domaine de l'unité de transport.

Les principes généraux et les règles d'instanciation par défaut identifiés pour ces algorithmes seront présentés dans la partie 2.3.3.

2.3.1.4. Cohérence d'un déploiement

Par construction, les déploiements produits automatiquement par les algorithmes d'instanciation évoqués précédemment se doivent d'être conformes par rapport à la définition des types domaines correspondants. Comme ces déploiements relèvent d'un certain nombre de choix par défaut, il importe de laisser à l'utilisateur la possibilité de les modifier afin de les personnaliser (par exemple, en ce qui concerne les décisions relatives aux cardinalités). Afin de maintenir la cohérence du modèle, la **conformité du déploiement par rapport aux types domaine** doit être assurée en cas d'une telle modification par l'utilisateur.

Dans le cadre d'une approche de prototypage rapide, un modèle fait l'objet d'une succession de boucles modélisation / exécution. Imposer à l'utilisateur de générer un nouveau déploiement s'il souhaite modifier les types domaine peut rapidement devenir un handicap à la fluidité et à la rapidité de cette démarche. Par conséquent, il importe de permettre la modification des types domaine même s'ils ont fait l'objet d'une instanciation, et de maintenir la cohérence des instances correspondantes suite à cette mise à jour.

Les méthodes identifiées pour le maintien de la cohérence d'un déploiement suite à une modification des types ou des instances domaine seront présentées dans la partie 2.3.4, ainsi que les choix adoptés dans le cadre d'IDEA.

2.3.2. Implémentation de la notion de type et instance domaine dans le DSL

2.3.2.1. Approche « classes/instances »

Pour implémenter la notion de type et instance domaine dans le DSL de description d'architecture, une première approche est de s'inspirer des langages à classe présentés dans la partie 1.2.2.2.

Le métamodèle sera ainsi structuré sur la base des notions *Classifier* et *InstanceSpecification* du métamodèle UML. Une même catégorie de constituants d'architecture (par exemple les Capacités) est alors représentée par deux concepts différents dans le métamodèle, un pour définir les types domaines (héritant de la notion de *Classifier*) et un pour les instances domaine (héritant de la notion de *InstanceSpecification*). A ces concepts sont également associées des spécialisations de *ValueSpecification* et *Slot*.

Le résultat de cette approche est illustré sur la Figure 17 sur l'exemple des Capacités : les Capacités de niveau type domaine sont définies par le concept *Capability*, les Capacités instances domaine par le concept *CapabilityInstance*.

Cette méthode introduit dans le métamodèle les notions de type et instance domaine de manière proche du standard (UML) et très structurée. Elle permet également de gérer aisément d'éventuelles différences de structures et de propriétés entre les concepts représentant les types et les instances domaine pour une même catégorie de constituants d'architecture.

En revanche, elle rend nécessaire la création de quatre concepts pour chaque catégorie de constituant d'architecture. Si ces catégories sont nombreuses (comme c'est le cas dans le domaine de Systèmes de Systèmes), le métamodèle résultant est complexe et donc difficile à maintenir et à appréhender.

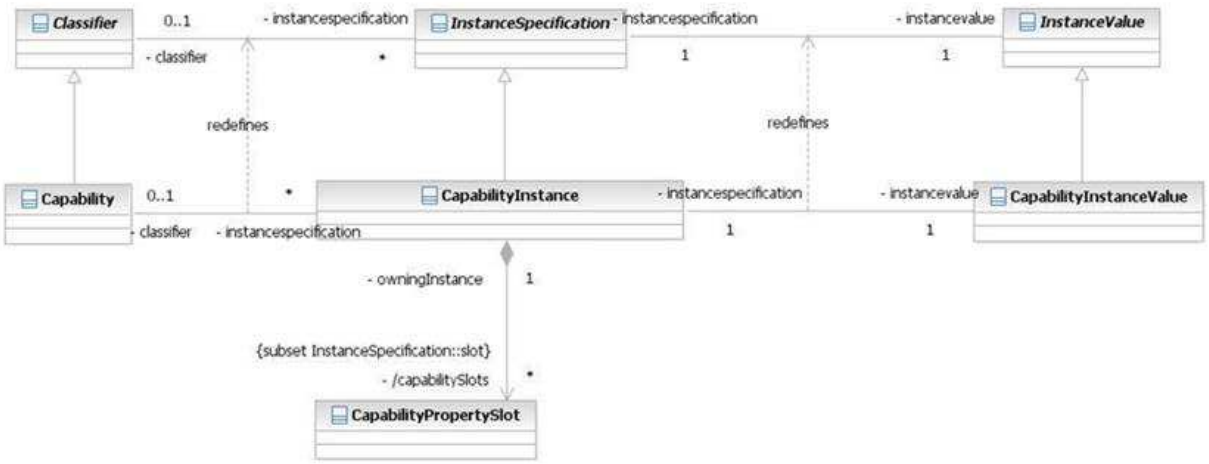


Figure 17 - Implémentation de la notion de *CapabilityInstance* dans une approche inspirée du métamodèle UML

2.3.2.2. Approche « prototype »

Pour implémenter les notions de type domaine et instance domaine dans le DSL, on peut également adopter une approche inspirée de la notion de prototype présentée dans la partie 1.2.2.2.

Les types domaine sont alors considérés comme des représentants caractéristiques d'une certaine catégorie de constituants, et servent de référence pour la création des instances domaine. Le mécanisme de clonage pose les bases de l'instanciation automatique évoquée en 2.3.1.3, en partant du principe qu'une instance domaine est une copie du type correspondant. Selon le principe de prototype, il n'y a besoin dans le métamodèle que d'un seul concept pour chaque catégorie de constituants d'architecture. Tout concept représentant un prototype comporte de plus une association vers lui-même pour représenter le lien entre les instances domaines et leur type. Cette association n'est pas inclue dans la notion de prototype, mais est nécessaire dans le cadre de notre langage de description d'architecture pour garder une traçabilité entre les instances domaines et leur type et permettre ainsi leur maintien en cohérence.

Le résultat de cette approche est illustré par la Figure 18 sur l'exemple des Capacités : les Capacités de niveau type domaine et celles de niveau instance domaine sont toutes issues du même concept *Capability*.

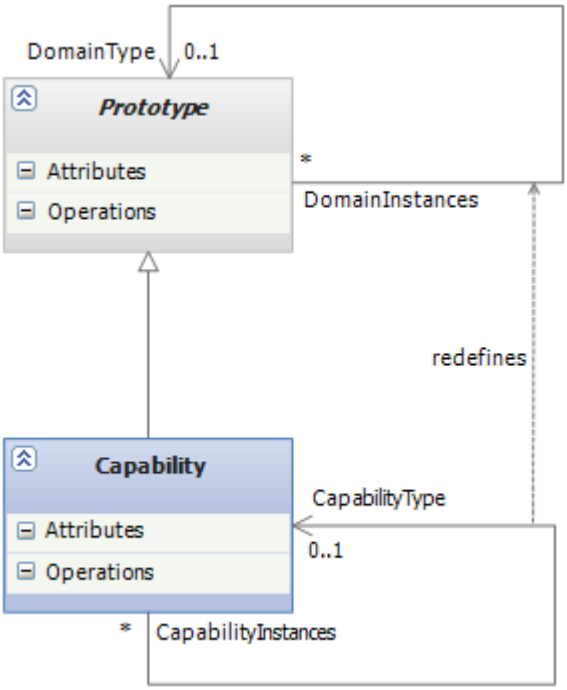


Figure 18 - Implémentation de la notion de *Capability* dans une approche inspirée prototypes

Cette approche permet d'introduire les notions de type et instance domaine dans le métamodèle de manière plus synthétique que l'approche « classe/instance », et rend donc le langage plus facilement compréhensible pour les différents intervenants des sessions d'architecture. Les modèles issus de ce langage, plus compacts, seront également plus faciles à maintenir (notamment en ce qui concerne le maintien de la cohérence).

On notera toutefois que les notions de type et instance domaine requièrent une certaine adaptation par rapport au principe de prototype. En particulier, l'évolution d'une instance domaine ne peut pas se faire de façon entièrement indépendante de son type, contrairement au principe usuel du clonage tel qu'exposé précédemment. Les modalités d'évolution d'une instance domaine sont en effet liées à la notion de cohérence vis-à-vis de son type[17].

2.3.2.3. Choix pour le DSL IDEA

Les travaux dans le domaine de la modélisation multiniveau ont également mené à l'élaboration de solutions différentes des deux approches initiales « classe / objet » et « prototype ».

En particulier, pour ne pas être contraints par la dualité classe / objet ni abandonner toute expression explicite de ces niveaux comme dans l'approche par prototypes, Atkinson & Kühne introduisent la notion de clabject [27, 28]. Un clabject représente indifféremment une classe ou un objet, et ses champs sont affectés d'un potentiel qui définit le nombre d'instanciations nécessaires avant que le champ correspondant ne requiert une valeur. Un champ de potentiel 1 équivaut ainsi à un attribut de classe. Si cette approche permet d'amener une grande flexibilité dans la définition de niveaux de modélisation pour un système informatique, pour l'implémentation de deux niveaux de modélisation seulement comme c'est le cas dans le métamodèle IDEA, cette notion n'apporte pas de différence majeure par rapport à l'approche classes / instances.

[17] Pour plus d'information sur les modalités d'évolution d'une instance domaine, se référer à la partie 2.3.4.

Entre les deux approches initiales, nous avons jugées prioritaires les considérations de compacité et de facilité de maintien de la cohérence, ainsi que la simplicité d'exploitation du langage résultant pour l'utilisateur. L'implémentation basée sur le mécanisme classes / objets a été ainsi été écarté car trop lourde au niveau du métamodèle (comparer Figure 17 et Figure 18), et d'une complexité théorique trop importance pour les utilisateurs (nécessité de manipuler des concepts différents selon le niveau de modélisation).

C'est donc l'approche « **Prototype** » qui a été adoptée, associée à un système de **contraintes** (règles de cohérence) pour assurer les relations de typage. Le principe de clonage pour l'automatisation de l'instanciation permet de plus de tirer profit du mécanisme générique de copier-coller offert par l'environnement DSL Tools.

2.3.3. Algorithmes d'instanciation

2.3.3.1. Périmètre de l'instanciation

Si l'on considère les instances domaine comme des copies des types domaine, il importe avant d'automatiser l'instanciation de définir et de cadrer la notion de copie. Dans une approche DSL, un élément de modèle se caractérise de la manière suivante :

- Le **concept** de métamodèle dont il est issu ;
- Les **valeurs** qu'il spécifie pour chacune des propriétés de ce concept ;
- Les **relations** dans lesquelles il est impliqué avec d'autres éléments.

Selon l'approche « Prototype » présentée précédemment, c'est un même concept de métamodèle qui définit les types et les instances domaines relatifs à une même catégorie de constituant d'architecture. Une instance domaine issue de la copie d'un type sera donc issue du même concept de métamodèle.

- Ex : Le type domaine représentant une unité blindée et l'instance domaine représentant l'unité blindée déployée « Unité blindée 1 » seront tous deux des éléments issus du concept de métamodèle *EnterpriseEntity*.

Les valeurs spécifiées par un type domaine pour les propriétés du concept dont il est issu représentent les spécificités du constituant qu'il représente par rapport à la catégorie générique. *A priori*, il n'y a aucune raison pour une instance domaine correspondant à ce type de redéfinir ces valeurs. Dans le cadre d'une instanciation automatique, une instance domaine issue de la copie d'un type spécifiera donc les mêmes valeurs de propriétés.

- Ex : Le concept *EnterpriseEntity* définit une propriété booléenne *IsMovable*, servant à indiquer si l'entité est mobile. Le type domaine représentant une unité blindée spécifie la valeur « vrai » pour cette propriété, et par défaut ce sera également le cas de toutes les instances domaines qui en seront issues.

Les stratégies à disposition d'un algorithme d'instanciation pour établir les relations entre les instances domaine peuvent être classées en trois catégories :

- **Instanciation par copie** – Le comportement d'instanciation est défini au niveau d'une association dans le métamodèle, et est le même pour toutes les relations issues de cette même association. On distingue trois comportements différents, illustrés sur la Figure 19 :

- Propagation de la copie au lien et à l'élément cible : les deux éléments impliqués dans la relation sont copiés, de même que la relation entre eux ;
- Propagation de la copie au lien seul : l'élément issu de la copie est relié au même élément que la source de la copie ;
- Pas de propagation de la copie : l'élément issu de la copie n'est pas impliqué dans une relation.

- **Instanciation a posteriori** – L'instanciation d'une relation établie entre deux types domaines et liée au contexte, c'est-à-dire à l'ensemble des éléments instanciés par l'algorithme.
- **Etablissement de relations sans type** – Il peut arriver qu'une association entre deux concepts du métamodèle définisse une relation métier qui n'a de sens qu'entre des éléments déployés. Une telle relation n'existe alors dans un modèle qu'entre des instances domaine[18], et n'a pas d'équivalent au niveau des types. Dans le métamodèle IDEA, c'est le cas de l'association *Plays* entre les concepts *Collaboration* et *EnterpriseEntity*. Dans certains cas, il est possible de construire de telles relations automatiquement à l'instanciation en les déduisant d'informations disponibles entre les types domaines.

Ces différentes stratégies peuvent être toutes appliquées dans le cadre d'un même langage. Afin de conserver une certaine cohérence dans le processus d'instanciation, on veillera à ce que la même stratégie soit appliquée à toutes les relations issues de la même association entre concepts de métamodèle.

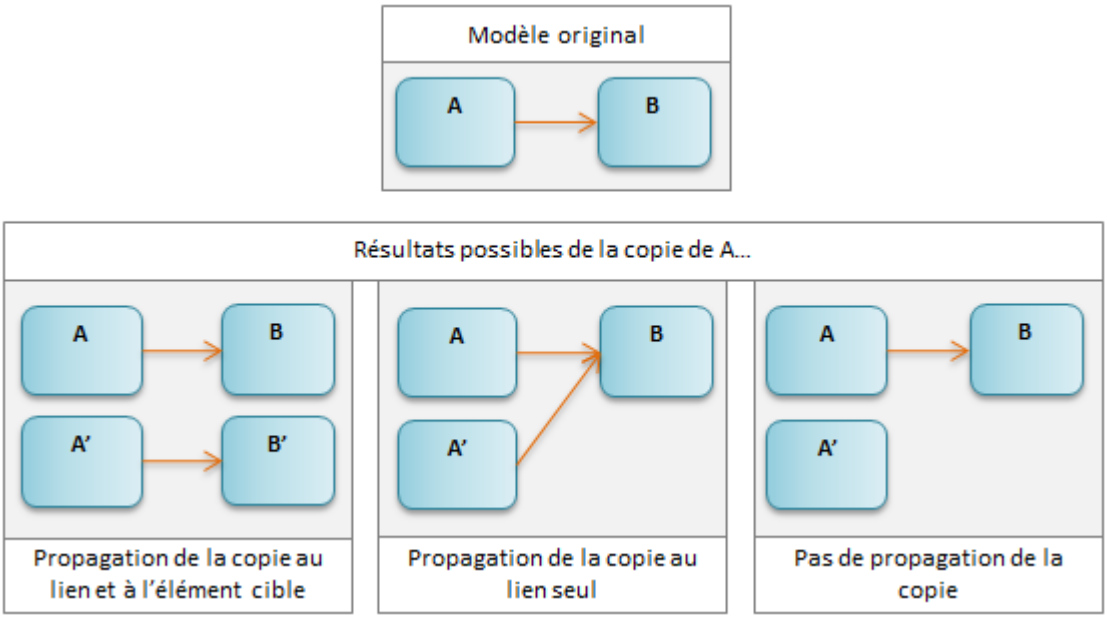


Figure 19 - Synthèse des comportements de copie pour les relations entre éléments de modèle

2.3.3.2. Instanciation par copie : relations hiérarchiques

Une relation hiérarchique exprime un couplage fort entre les deux éléments qu'elle relie, et traduit le fait que l'élément cible fait partie intégrante de l'élément source. Tous les éléments liés par une

[18] Le cas d'une relation métier qui n'aurait de sens qu'entre des éléments types peut également arriver, mais présente peu d'intérêt pour ce chapitre : en effet, son comportement pour l'instanciation sera trivialement de ne pas propager la copie.

relation hiérarchique à un type domaine doivent être instanciés avec lui, pour que l'instance domaine résultante soit dotée de la même décomposition.

- Ex : A l'instanciation d'un type domaine définissant un véhicule doivent être également instanciés tous les équipements embarqués sur ce véhicule.

L'objectif principal de la définition des types domaine étant la capitalisation, un même type domaine est susceptible d'être cible de plusieurs relations hiérarchiques. Une instance domaine, quant à elle, ne peut être cible que d'une seule relation hiérarchique.

- Ex : dans le modèle de convoi, les types de véhicules « Blindé » et « Camion » peuvent être équipés (relation *Contains*) du même type de « Radio », mais plusieurs véhicules physiques ne peuvent pas partager le même équipement radio.

L'instanciation de deux types domaine source d'une relation hiérarchique vers le même élément donnera donc lieu à la création de deux instances domaine de ce dernier, une dans chacune des instances domaine correspondants aux deux types sources.

Dans cette même optique de capitalisation, les relations hiérarchiques sont susceptibles dans le métamodèle d'être dotées d'une propriété de type *multiplicité*. Pour une relation entre types domaines dans un modèle, la valeur spécifiée pour cette propriété définit une contrainte sur le nombre d'éléments à déployer, et donc d'instances domaine attendues.

- Ex : dans le modèle de convoi, la relation hiérarchique entre le type « Convoi » et le type « Unité de transport » est dotée d'une propriété *multiplicité*, dont la valeur vaut (2..4).

Dans le cas d'une cardinalité (n..m) avec n ≠m, la détermination du nombre exact d'instances à générer est un choix d'architecture déterminant ne pouvant pas être pris définitivement par un algorithme. En revanche, il importe pour celui-ci de produire un déploiement par défaut en accord avec la cardinalité définie, afin de fournir à l'utilisateur une structure de base qu'il pourra alors retoucher. Par défaut, l'algorithme instanciera donc le nombre minimum d'éléments, soit la valeur du minimum de la cardinalité[19].

Le comportement proposé pour l'instanciation des relations hiérarchiques est ainsi la propagation de la copie au lien et à l'élément cible, contrainte par une éventuelle valeur de *multiplicité* spécifiée sur la relation. Il est synthétisé dans la Figure 20.

Cette méthode a été implémentée pour le DSL IDEA dans les algorithmes d'instanciation des *EnterpriseEntity* (pour les relations *Aggregates* et *Contains*) et des *Roles* (pour les relations *Involves*).

[19] En particulier, cela signifie que si une relation hiérarchique est dotée d'une propriété *multiplicité* dont la valeur est de type (0..m), aucune instance domaine de l'élément cible n'est créée.

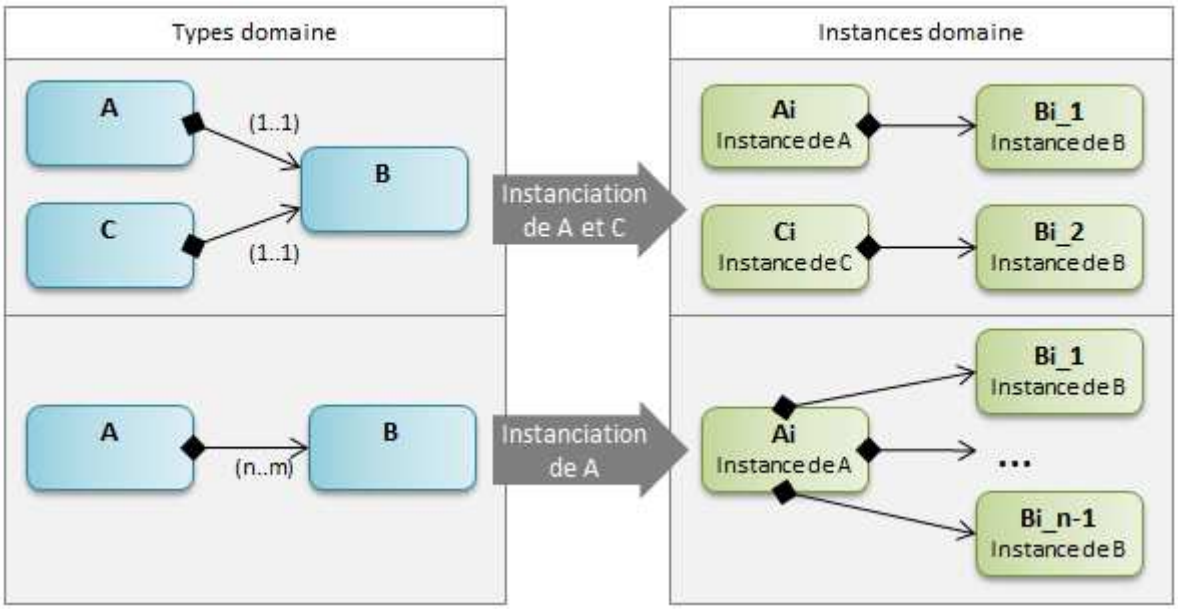


Figure 20 - Principes de l'instanciation des relations hiérarchiques

2.3.3.3. Instanciation a posteriori : relations transverses à une organisation hiérarchique

Dans le cas des relations transverses établies entre des éléments appartenant à la même organisation hiérarchique, on considère que l'instanciation de l'élément source ne doit pas impliquer automatiquement l'instanciation de l'élément cible. En revanche, dans le cas où deux éléments types domaines d'une même organisation hiérarchique et liés par une telle relation ont été instanciés, la relation doit être recréée au niveau des instances domaine concernées.

Du point de vue de l'algorithme d'instanciation, cela signifie que ces relations doivent être établies selon le schéma défini par les types domaines une fois que l'organisation hiérarchique complète a été instanciée. Une relation transverse ne porte pas de comportement d'instanciation propre, mais est subordonnée à l'instanciation de la structure hiérarchique dans le contexte de laquelle elle est établie. Ce principe est illustré par la Figure 21, sur lequel la relation transverse figure en pointillés par opposition aux relations hiérarchiques en trait plein.

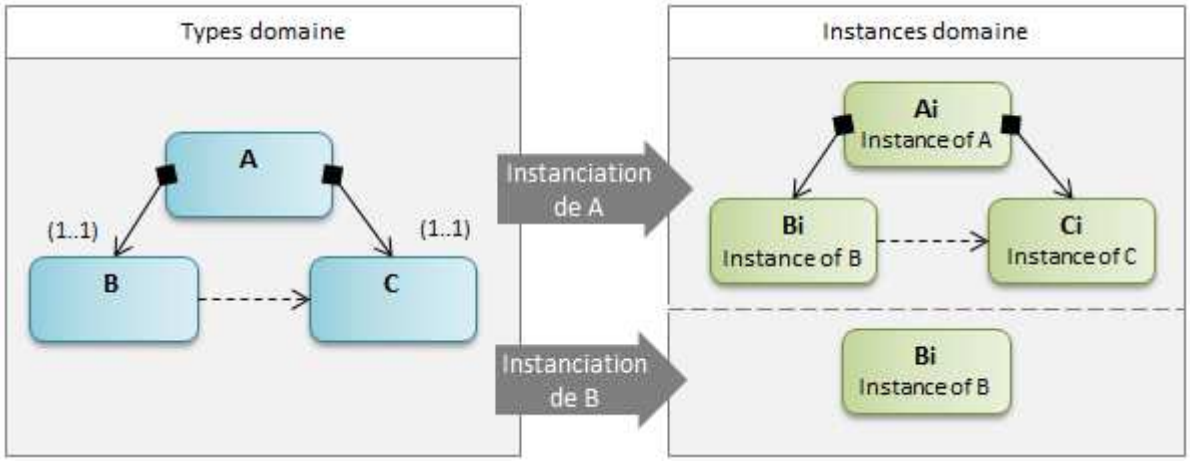


Figure 21 – Principes de l'instanciation des relations transverses (1)

De même que les relations hiérarchiques, les relations transverses peuvent être dotées d'une propriété de type *Multiplicité*. Si l'on considère une telle relation établie au niveau type domaine entre des éléments cibles de relations hiérarchiques elles-mêmes dotées d'une propriété *Multiplicité*, trois cas sont à prendre en compte.

Pour les développer, nous considèrerons au niveau des types domaine un élément A, source de deux relations hiérarchiques ciblant respectivement un élément B avec une *multiplicité* (1..1) et un élément *C* avec une *multiplicité* (n..m). B est source d'une relation transverse ciblant C avec une *multiplicité* (n'..m'). Suivant le principe d'instanciation des relations hiérarchiques établi précédemment, à l'instanciation de A seront créées n instances domaines de C. Cette situation est illustrée dans la Figure 22.

- Si n appartient à l'intervalle [n'..m'], alors le nombre d'instances domaine de C est compatible de la *multiplicité* de la relation transverse. On peut alors instancier une relation transverse entre l'instance de B et chacune des instances de C.
  Cette situation correspond au Cas 1 de la Figure 22.
- Si n est supérieur à m', alors il y a plus d'instances domaine de C qu'il n'est possible d'établir de relations transverses depuis l'instance de B. Dans ce cas, l'algorithme d'instanciation devra faire un choix par défaut pour déterminer quelles instances de C relier à l'instance de B. Dans le cadre du DSL IDEA, nous avons choisi les m' premières instances par ordre chronologique de création.
  On notera que dans le cas où la *multiplicité* de la relation hiérarchique ciblant B au niveau des types domaine est différente de (1..1), une décision par défaut est également à prévoir dans l'algorithme pour répartir les différentes instances de C aux instances de B.
  Cette situation correspond au Cas 2 de la Figure 22.
- Si n est inférieur à n', alors il y a moins d'instances domaine de C qu'il ne faut au minimum de relations transverses depuis l'instance de B. A moins de contredire l'hypothèse selon laquelle il n'y a pas de propagation de l'instanciation pour les relations transverses, il n'existe pas de possibilité pour l'algorithme d'instanciation de générer un déploiement conforme aux types domaine. Il est possible de détecter ce cas directement au niveau des types domaine, et de les invalider pour interdire leur instanciation. Pour le DSL IDEA, nous avons choisi une solution plus souple, à savoir instancier des relations transverses entre l'instance de B et toutes les instances de C, et invalider l'instance de B afin d'indiquer l'erreur pour correction ultérieure par l'utilisateur.
  Cette situation correspond au Cas 3 de la Figure 22.

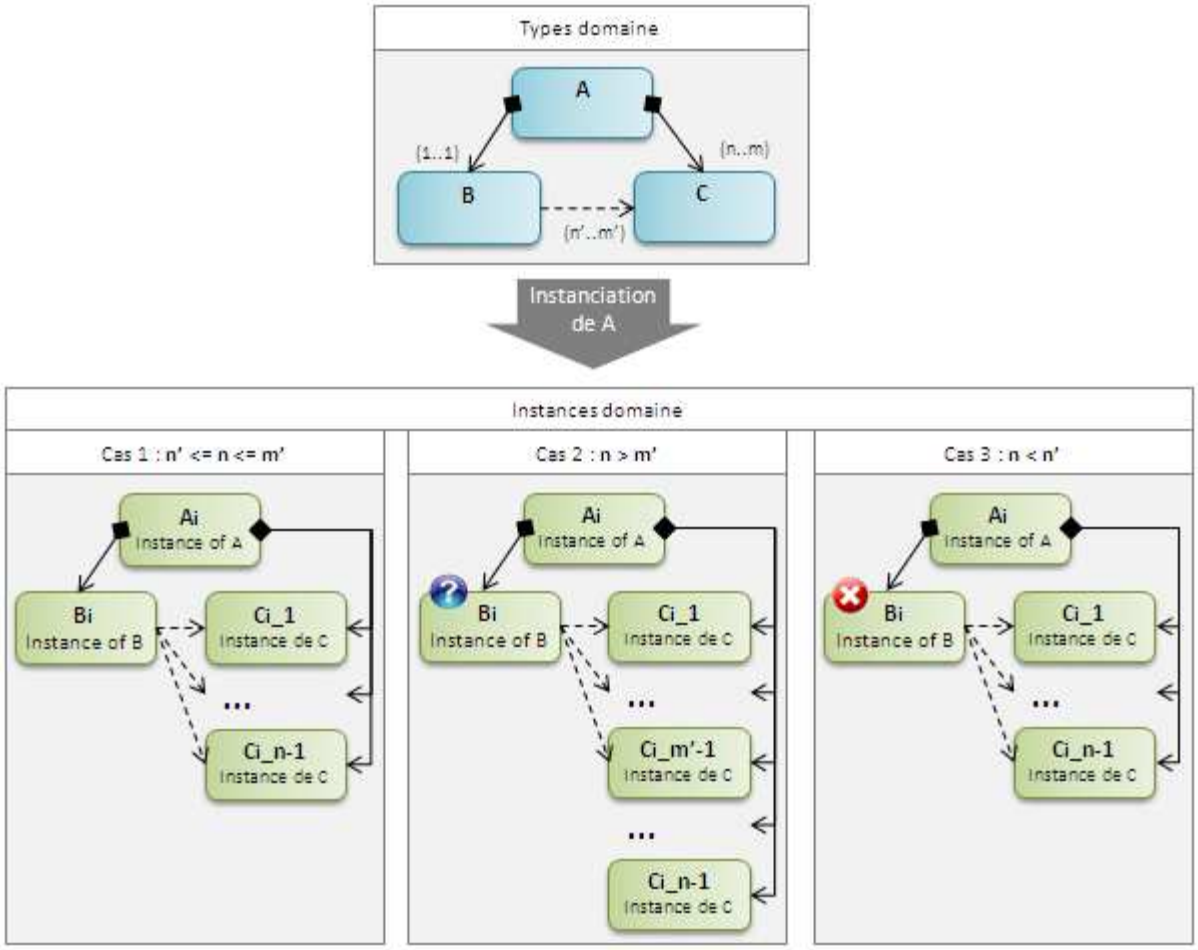


Figure 22 - Principes de l'instanciation des relations transverses (2)

Des situations similaires peuvent survenir dans le cas où le nombre de relations transverses dans lesquelles peut être impliqué un élément de modèle est contraint par les cardinalités définies au niveau du métamodèle. Par exemple, c'est le cas des interactions de service entre les Collaborations d'un modèle IDEA. Pour éviter toute ambiguïté quant au service à appeler lors de l'exécution, un Rôle instance domaine consommant un service ne peut être lié qu'à un seul fournisseur. En revanche un Rôle instance domaine peut fournir un même service au bénéfice de plusieurs consommateurs.

Afin d'illustrer ce propos, considérons comme représenté sur la Figure 23 le cas d'une Collaboration de niveau type domaine, contenant deux Rôles liés entre eux par une interaction de service. La Collaboration définit une *multiplicité* (n..m) avec n>1 pour l'un des Rôles et (1..1) pour l'autre. Lorsque le Rôle concerné par la *multiplicité* (n..m) est le consommateur de service, il est donc possible pour l'instance du fournisseur de fournir le service aux n instances domaines. Mais lorsque le Rôle concerné par la *multiplicité* (n..m) est le fournisseur de service, il faut à l'algorithme choisir pour l'instance du consommateur un fournisseur par défaut. En l'occurrence, ce sera le premier dans l'ordre chronologique de création. Il devra par la suite être possible pour l'utilisateur de changer manuellement le fournisseur d'une telle relation.

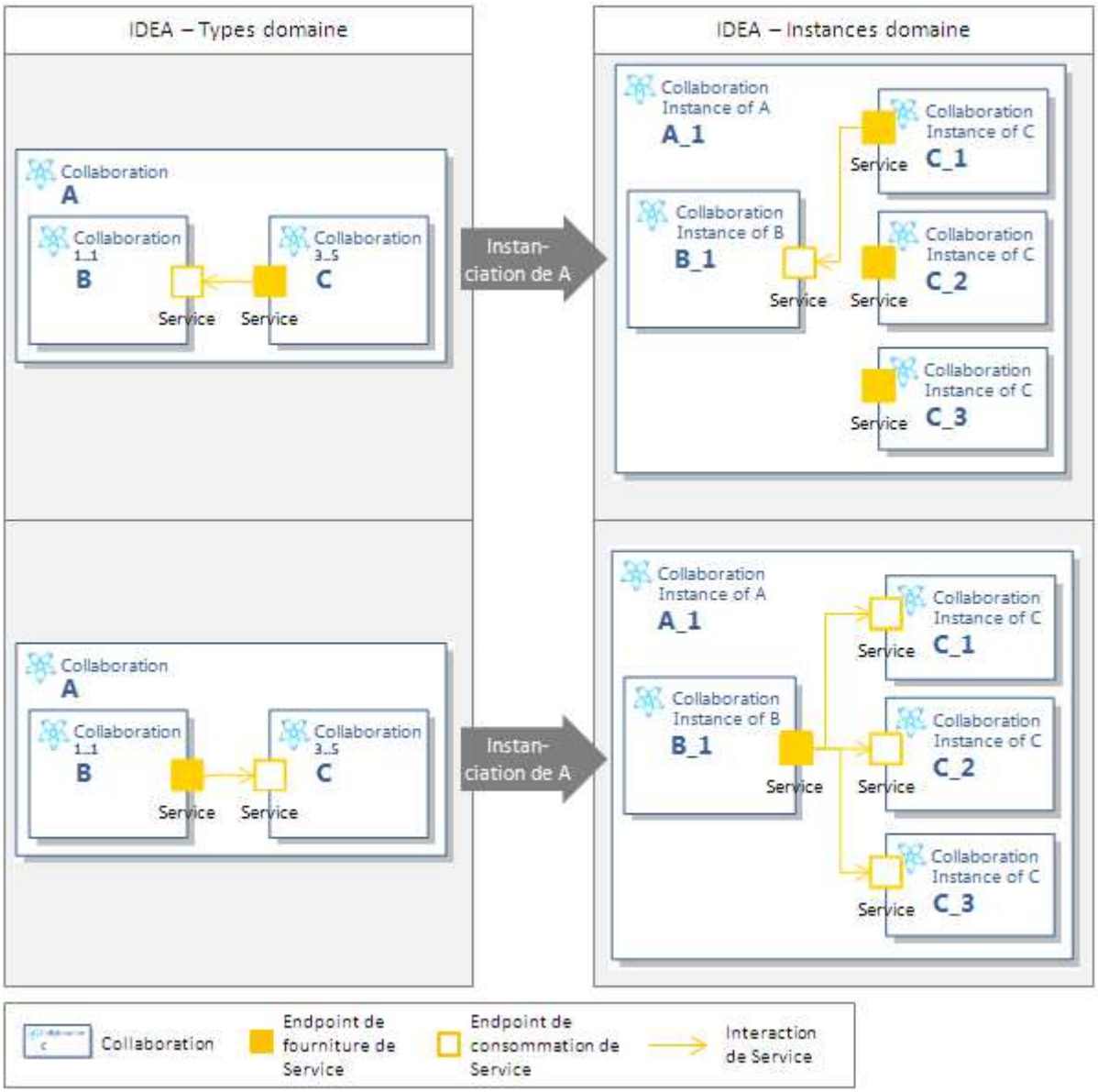


Figure 23 - Instanciation des interactions de service (assemblage de captures d'écran d'un modèle IDEA)

2.3.3.4. Etablissement de relations sans type : instanciation conjointe COI/Collaboration

Les principes et méthodes énoncés dans les deux parties précédentes permettent d'instancier, séparément, un ensemble de *Collaborations* et un ensemble d'*EnterpriseEntity*. Pour que le modèle soit complet et exécutable, il reste à établir des relations *Plays* entre les deux ensembles. D'un point de vue métier, une telle relation exprime l'allocation de ressources à un rôle déployé et n'a donc de sens qu'entre des instances domaine.

Une même *EnterpriseEntity* peut répondre à plusieurs missions différentes, dans le contexte de différentes *Collaborations*, et ce au sein d'un même déploiement. Le choix de l'allocation des *Collaborations* aux *EnterpriseEntity* répond donc à des critères métiers relevant entièrement de l'architecture et/ou de la doctrine, et ne peut pas être automatisé. Cependant, et en particulier si l'organisation est complexe, la création manuelle de ces relations peut rapidement devenir une tâche laborieuse et monotone pour l'utilisateur.

Le concept de *COI* échappe cependant à cette règle puisqu'un tel regroupement d'*EnterpriseEntity* est dédié à un objectif particulier, modélisé par la *Collaboration* qui lui est associée par la relation *Implements*. Cette *Collaboration* définit le mode d'action et d'interaction mis en œuvre par les membres de la COI pour atteindre son objectif (cf. 2.2.2.7). Par conséquent, il apparait souhaitable qu'à l'instanciation d'une *COI* soit également instanciée la *Collaboration* qu'elle implémente, et qu'une relation *Plays* soit établie entre leurs instances domaine. Les *Roles* résultant de l'instanciation de la *Collaboration* doivent alors être alloués aux entités instances domaine participants à la *COI* ou à leurs descendantes, sous réserve qu'elles aient les capacités de les jouer. Ce principe est illustré par la Figure 24.

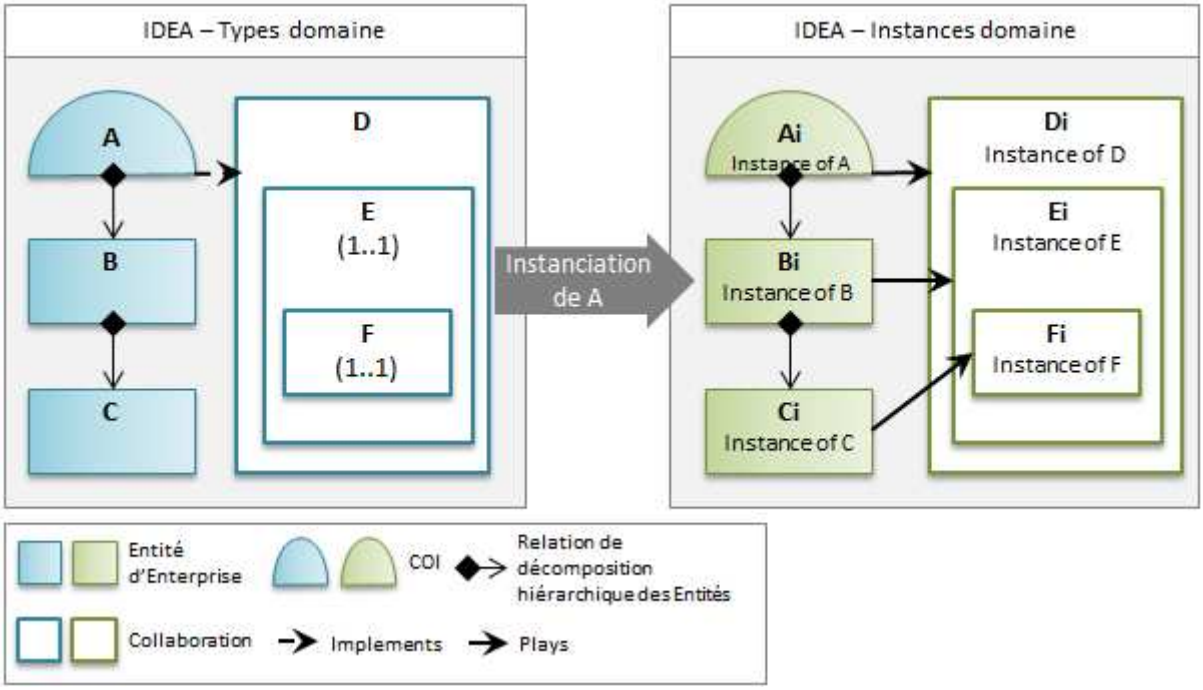


Figure 24 – Vue schématique du résultat attendu de l'instanciation conjointe COI/Collaboration

Si l'on considère comme déjà instanciées d'un côté la *COI*, avec tous ses participants et tous leurs descendants, et d'un autre la *Collaboration* implémentée, avec tous ses *Roles* et leurs descendants. La cohérence de l'établissement des relations *Plays* entre les instances domaines des entités et des rôles présuppose alors de savoir répondre à deux types de problématiques : l'estimation de la capacité d'une *EnterpriseEntity* à jouer un *Role*, et la gestion du cas où plusieurs membres de la COI peuvent jouer un même *Role*.

- Les exigences envers un acteur potentiel sont modélisées par la relation entre les concepts *Collaboration* et *CapabilityCell*. Pour pouvoir jouer une *Collaboration*, une *EnterpriseEntity* doit donc être dotée de l'ensemble des *Capability* et pouvoir fournir l'ensemble des *Service* listés dans la ou les *CapabilityCell* de cette *Collaboration*.
- Considérons à présent le cas d'un Rôle contenu dans la *Collaboration* implémentée par la COI ou descendant d'un tel Rôle, et pouvant être joué par plusieurs *EnterpriseEntity* participant à la *COI* ou descendant de participants à la *COI*. La relation *Implements* ayant une signification métier forte, on cherchera avant tout à allouer la *Collaboration* concernée à une éventuelle *COI* qui l'implémenterait. A noter que dans ce cas, on restreindra l'allocation des *Collaboration* descendantes de la *Collaboration* concernée aux *EnterpriseEntity* descendant de cette *COI* au lieu

de prendre en compte tous les descendants de la *COI* parente. Si aucun des candidats n'est une *COI* implémentant la *Collaboration* concernée, alors on privilégiera les participants directs de la *COI* par rapport à leurs descendants.

Par ailleurs, dans le cas où le rôle est défini au niveau type domaine avec une *multiplicité* (n..m) où n>1, il faut par ailleurs allouer une instance domaine de cette *Collaboration* à chaque *EnterpriseEntity* instance candidate, dans les limites de cette *multiplicité*.

2.3.3.5. Mise en œuvre dans le DSL IDEA

L'instanciation automatique des rôles et des entités d'entreprise dans le DSL IDEA est gérée principalement par deux méthodes `Instantiate()`, liées aux concepts *Collaboration* et *EnterpriseEntity*. Leur appel se fait par l'intermédiaire d'un menu contextuel dédié, accessible à l'utilisateur sur les éléments de modèle de niveau type domaine et issus de ces concepts. Le principe général de ces méthodes est le même :

- 1 – Création et initialisation de l'instance par copie des propriétés, établissement du lien de traçabilité ver le type domaine ;
- 2 – Instanciation de l'organisation hiérarchique issue du type domaine, par appel récursif ;
- 3 – Etablissement des relations transverses entre les instances domaine créées à l'étape précédente.

Lorsque la méthode de la classe *EnterpriseEntity* est appelée sur une *COI*, elle appelle en sus l'instanciation de la *Collaboration* implémentée par cette *COI*. Elle applique ensuite les principes énoncés en 2.3.3.4 pour établir les relations *Plays*.

2.3.4. Cohérence d'un déploiement

Par définition, les types domaine représentent une spécification des déploiements possibles pour les instances domaines correspondantes. Les déploiements par défaut produits par les algorithmes d'instanciation présentés à la partie précédente respectent dans la plupart des cas, par construction, la conformité des instances domaines par rapport aux types domaine[20]. Ces algorithmes effectuant parfois des choix par défaut, il est nécessaire de permettre à l'utilisateur de modifier les déploiements. Il est donc indispensable de mettre en place des règles automatiques permettant de s'assurer à tout moment de cette conformité.

On rappelle par ailleurs que dans le cadre d'une approche de prototypage rapide, un modèle est susceptible de faire l'objet d'une succession de boucles modélisation / exécution. Imposer à l'utilisateur de générer un nouveau déploiement s'il souhaite modifier les types domaine peut rapidement devenir un handicap à la fluidité et à la rapidité de cette démarche. Par conséquent, il importe de permettre la modification des types domaine même s'ils ont fait l'objet d'une instanciation, et maintenir la cohérence des instances domaine suite à cette mise à jour.

Cette partie présentera les méthodes et moyens mis en œuvre pour maintenir la cohérence d'un déploiement vis-à-vis de sa spécification au niveau des types domaine, quels que soient les éléments de modèle faisant l'objet d'une modification de l'utilisateur.

[20] A l'exception du cas des relations transverses évoqué en 2.3.3.3.

2.3.4.1. Modification des instances domaine

Le maintien en cohérence d'un déploiement à la modification des *instances* peut se mettre en œuvre de deux manières :

- **Approche 1** – Les seules modifications autorisées à l'utilisateur sur le déploiement sont celles qui sont **en accord avec la spécification des types domaines**.

  La cohérence du déploiement, et donc l'exécutabilité du modèle, sont ainsi assurées en permanence. Cette méthode nécessite un grand nombre de règles de cohérence, chaque action possible devant déclencher une règle qui évaluera l'impact sur la cohérence du déploiement. Selon le cas, la règle autorisera ou interdira l'action. L'établissement d'une règle de cohérence pour chaque type de modification peut toutefois se révéler une action lourde à mettre en œuvre, en termes de complexité du code du DSL mais aussi de temps d'exécution[21].

  Par exemple, suite au déploiement d'une « Unité blindée », il serait interdit à l'utilisateur de supprimer l'un des fantassins du groupe de combat. En effet, cela impliquerait que la cardinalité effective de ses membres devienne incompatible avec celle définie pour le *type* « Groupe de combat INF ».

- **Approche 2** – L'ensemble des modifications **en accord avec le métamodèle** sont autorisées sur le déploiement, mais des règles de **validation** informent d'une éventuelle contradiction avec les types domaine.

  Cette approche présente l'avantage de laisser à l'utilisateur toute latitude dans la personnalisation de son déploiement, en laissant à sa charge le choix de mettre à jour ses types domaine ou de modifier ses instances domaine selon la manière dont il souhaite corriger l'incohérence. En revanche, l'exécutabilité du modèle n'est pas maintenue automatiquement, et l'utilisateur doit déclencher manuellement une validation s'il souhaite connaître l'impact de ses modifications sur la cohérence du déploiement.

  Par exemple, dans le cas d'un déploiement d'« Unité blindée », l'utilisateur pourrait supprimer l'un des fantassins, mais l'instance de « Groupe de combat INF » présenterait alors une erreur de validation précisant l'incohérence de la cardinalité.

Dans le cadre du DSL IDEA, c'est l'approche 2 qui a été choisie de par la liberté qu'elle accorde à l'utilisateur. En effet, les restrictions imposées dans le cadre de l'approche 1 sont trop limitantes, et peuvent constituer un obstacle au bon déroulement de la modélisation des instances domaine. Par exemple, l'interdiction de supprimer ou d'ajouter un élément dans une organisation hiérarchique si cela n'est pas en accord avec la *Multiplicité* prévue rend impossible tout remplacement d'un de ces éléments par un autre. Or un tel remplacement est une action courante et indispensable dans le cadre d'un déploiement.

Cependant, comme l'approche adoptée ne garantit pas automatiquement la cohérence du déploiement, nous avons choisi de limiter le périmètre des modifications autorisées à l'utilisateur afin de maîtriser au mieux l'impact de ses actions. La liste des erreurs levées par les règles de validation mises en place pour vérifier la cohérence d'un déploiement est présentée en Annexe 2.

[21] En particulier par le fait que les règles de cohérence se déclenchent en cascade.

2.3.4.2. Modification des types domaine

Le maintien en cohérence d'un déploiement à la modification des types domaine peut se mettre en œuvre de deux manières :

- **Approche 1** – Toute modification des types domaine **est propagée** aux instances correspondantes, en se conformant aux mêmes principes que celles des algorithmes d'instanciation.

  De même que pour l'approche 1 de la partie précédente, cette méthode assure en permanence la cohérence du déploiement et l'exécutabilité du modèle. Elle requiert également un recours intensif aux règles de cohérence, dont la tâche sera pour la plupart d'effectuer au niveau des instances domaine des actions similaires à celles réalisées sur les types.

  Par exemple, considérons un « Blindé » de niveau type, ayant été instancié. Si on supprime l'arme « Mitrailleuse » embarquée sur le « Blindé », pour chaque *instance* correspondante l'*instance* de « Mitrailleuse » sera automatiquement supprimée.

- **Approche 2** – Les modifications réalisées sur les types domaine **ne sont pas propagées** aux instances correspondantes, mais des règles de **validation** informent d'une éventuelle incohérence.

  Cette approche permet de ne pas impacter d'éventuelles modifications effectuées par l'utilisateur sur le déploiement, et d'éviter la lourdeur induite par le déclenchement de règles de cohérences à chaque action utilisateur.

  Par exemple, dans l'exemple de la suppression de la « Mitrailleuse » sur un « Blindé » de niveau type, aucune instance ne sera impactée. En revanche, les Instances correspondantes présenteront une erreur de validation.

Pour le DSL IDEA, la souplesse de modélisation permise par l'approche 1 de propagation des modifications des types domaine aux instances correspondantes a été jugée essentielle pour les utilisateurs. En effet, en évitant de devoir retoucher les déploiements à chaque mise à jour des types domaine, elle optimise la faisabilité des boucles de modélisation/simulation. La liste des règles de cohérence mises en place pour assurer cette propagation est présentée en Annexe 3.

2.4. Modélisation de configurations dans l'entreprise

2.4.1. Analyse du besoin

2.4.1.1. Notions de configuration et de reconfiguration d'un élément de modèle

La configuration d'un constituant d'architecture désigne l'ensemble des sous-constituants qui le composent. Par analogie, on utilisera le terme de **configuration d'un élément de modèle** pour désigner l'ensemble des éléments de modèle qui le composent.

En particulier, on s'intéressera aux configurations des entités d'entreprises et des rôles selon les définitions suivantes :

- La configuration d'une *EnterpriseEntity* est l'ensemble des entités qui la composent au sens des relations de décomposition *Contains*, *Aggregates*, *Hosts* ou *Gathers*.

  Par exemple, l'entité « Blindé » du modèle présenté en 2.1 présente deux configurations, *base* et *commandement*, dont la décomposition en termes d'équipements est illustrée par la Figure 25.

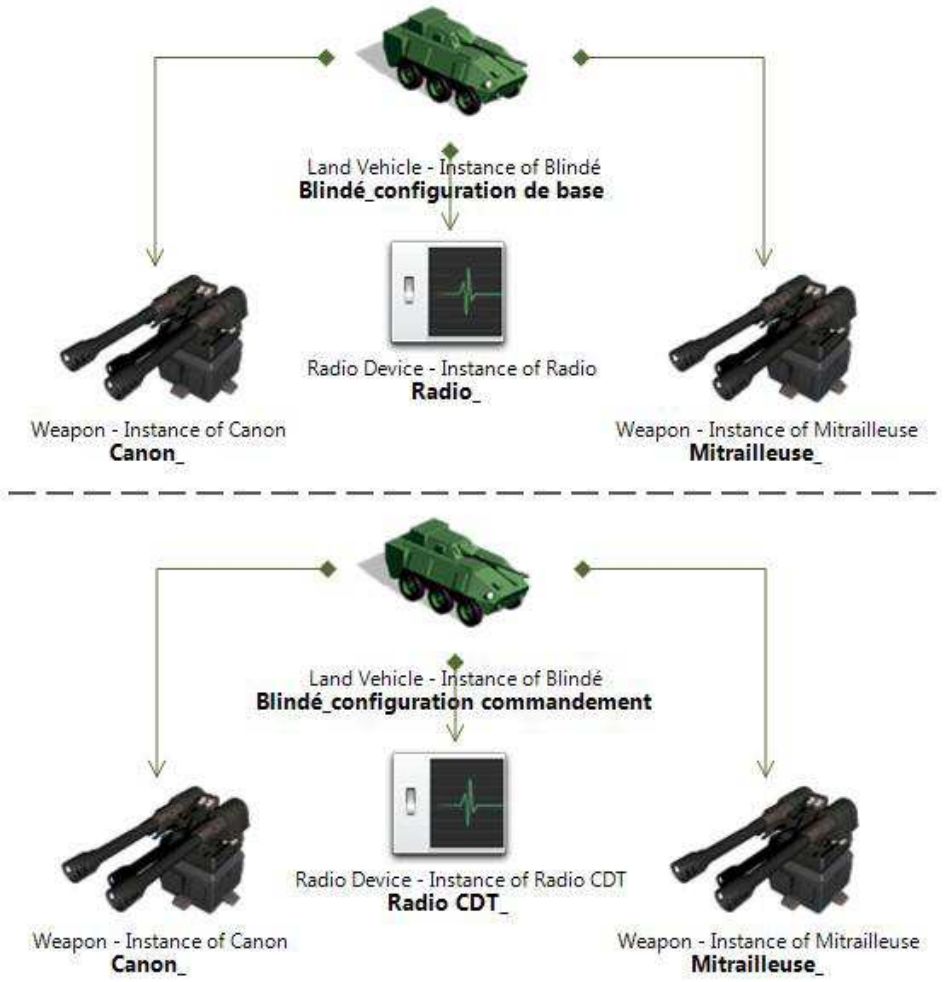


Figure 25 - Composition de l'entité "Blindé" dans ses deux configurations

- La configuration d'un *Role* est l'ensemble des rôles qu'il contient au sens des relations *Involves*, ainsi que les interactions qu'il définit entre eux.

  Par exemple, le rôle « Protection du convoi » du modèle présenté en 2.1 présente trois configurations, nominale, combat et combat avec appui, dont la décomposition en termes de rôles contenus est illustrée par la Figure 26.

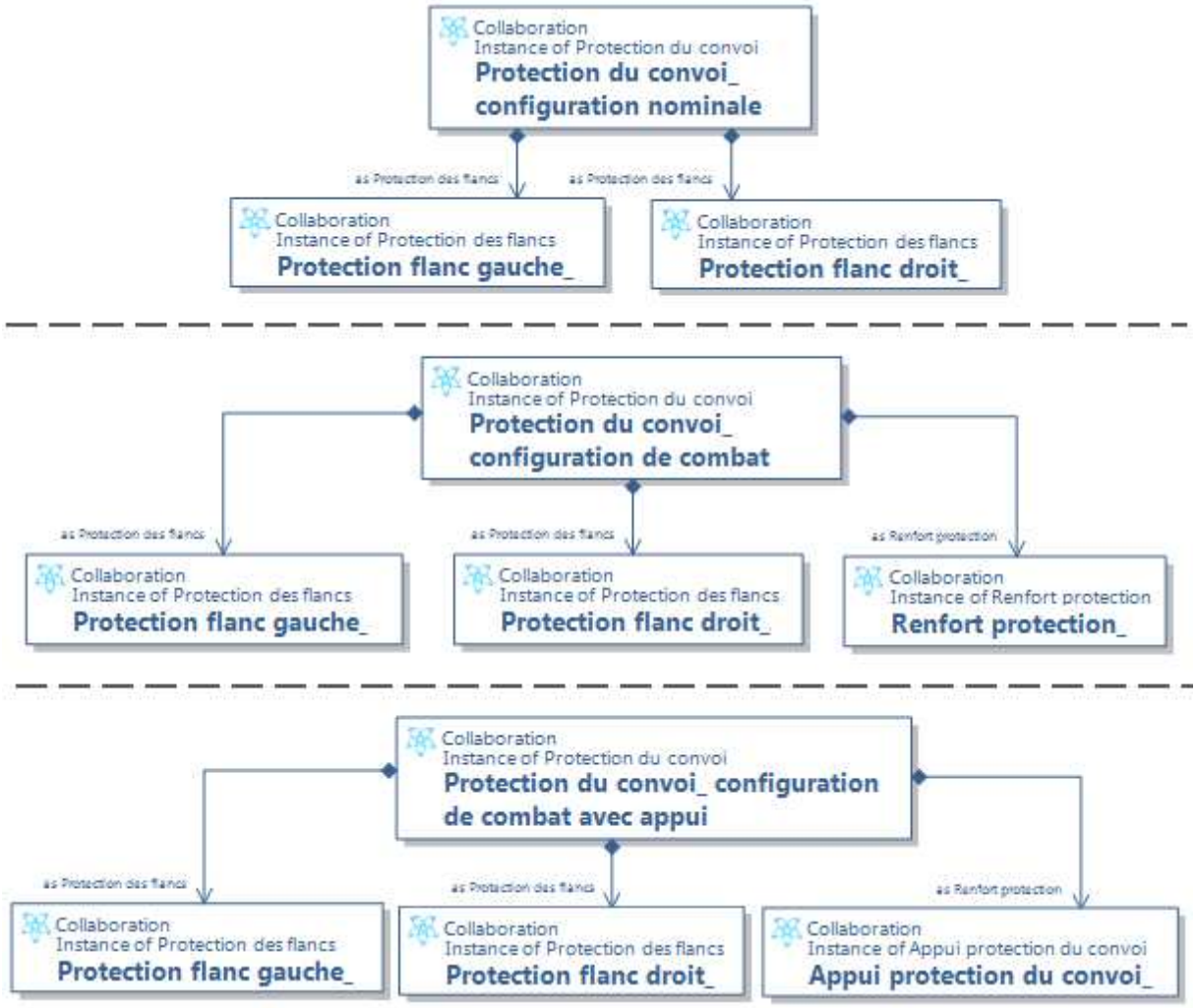


Figure 26 – Composition du rôle "Protection du convoi" dans ses trois configurations

Chercher à représenter deux configurations différentes pour un même constituant d'architecture ne relève pas de la même intention que de représenter deux constituants d'architecture différents. Par conséquent, la démarche de modélisation ne doit pas être la même. Dans le modèle IDEA, nous considérons que l'ensemble des configurations possibles pour un même constituant d'architecture est à définir au niveau du type domaine modélisant ce constituant.

Configurer un élément de niveau **type domaine** revient alors à **instancier une configuration en particulier**. Ainsi, les deux configurations du « Blindé » sont à modéliser au niveau du type domaine correspondant et les deux blindés configurés représentés sur la Figure 25 sont des instances domaine issues du même type. De même, les deux rôles de protection de la Figure 26 sont des instances du même rôle type « Protection du convoi ».

Reconfigurer un constituant d'architecture signifie le faire **passer d'une configuration à une autre**. Ainsi, reconfigurer une **instance domaine** revient à modifier l'ensemble des éléments de modèle qui le composent. Par exemple, reconfigurer un blindé de sa configuration de base vers la configuration de commandement revient à remplacer sa « Radio » par une « Radio CDT ».

2.4.1.2. Principes de modélisation des configurations

Dans certains cas, il est souhaitable que l'ensemble des configurations possibles pour un constituant soit déterminé et non extensible, c'est-à-dire strictement restreint à un certain nombre de configurations clairement identifiées et caractérisées. La définition de ces configurations possibles fait alors partie intégrante de la définition du constituant, au même titre que sa décomposition hiérarchique ou toute autre caractéristique d'architecture. Par exemple, définir les configurations du « Blindé » revient à exprimer le fait qu'il peut embarquer une « Radio » ou une « Radio CDT », mais aucun autre type de moyen de communication.

Dans la suite du document, on fera référence à ce principe en utilisant le terme **configurations de périmètre fermé**. La mise en œuvre de ce principe pour décrire les configurations des entités d'entreprise sera présentée en 2.4.2.

Dans d'autres cas, la définition d'une configuration pour un constituant n'a pas pour objectif de déterminer la nature des sous-éléments, mais d'exprimer à leur égard des attentes en termes de caractéristiques. On ne cherche alors pas à modéliser l'ensemble des configurations possibles pour ce constituant de manière explicite, d'autant plus qu'il est potentiellement infini. Par exemple, on peut définir les configurations du rôle de « Protection du convoi » en considérant simplement que dans une situation de combat, il doit contenir un rôle devant être capable d'envoyer des données d'observation, de prendre en compte une situation tactique (notée SiTac), d'échanger des ordres et des comptes-rendus et (surtout) de fournir un service d' « Appui protection » aux rôles en charge de la protection des flancs. Il n'y a pas de préjugé sur la nature du rôle (« Renfort protection » ou « Appui protection du convoi »), seul importe le fait qu'il soit capable d'interagir de la manière spécifiée. Cette définition regroupe ainsi les deux configurations de combat avec et sans appui présentées sur la Figure 26.

Dans la suite du document, on fera référence à ce principe en utilisant le terme **configurations de périmètre libre**.

Dans notre DSL, cela correspond à exprimer au niveau d'un type domaine un certain nombre de caractéristiques attendues pour les éléments composant les instances domaines qui en dérivent, sans se préoccuper de la manière dont ils disposent de ces caractéristiques. C'est ainsi que nous avons choisi de traiter les configurations des rôles / collaborations, qui fera l'objet de la partie 2.4.3.

2.4.2. Configurations des entités d'entreprises

2.4.2.1. Variabilité

La définition des configurations de périmètre fermé pour les entités d'entreprise peut être appréhendée de manière similaire aux approches de modélisation de la variabilité.

Initialement étudiée dans le cadre des lignes de produits logiciels, la notion de variabilité est désormais un objet d'étude dans des domaines variés, concernant des systèmes logiciels aussi bien que physiques [21]. Sur le principe, modéliser les variations pour une ligne de produits revient à

exprimer la *commonalité* et la *variabilité* dans la définition d'un ensemble d'éléments de la même famille. Par rapport aux configurations des entités d'entreprise dans notre DSL, cela correspond à exprimer au niveau d'un type domaine les points communs et les aspects variables en termes de structure pour l'ensemble des instances domaine qui en dérivent.

Les travaux actuels dans le domaine de la variabilité pour les DSL préconisent pour la plupart de séparer le métamodèle décrivant les aspects de variation et le métamodèle décrivant les aspects liés au domaine lui-même [22][23][24]. Dans le cas des configurations d'entités d'entreprise, nous considérons que la variabilité n'est pas un besoin annexe au métier, mais au contraire une caractéristique opérationnelle à part entière. La variation fait alors partie intégrante du domaine lui-même, et les mécanismes permettant son expression seront définis directement dans le métamodèle IDEA.

En particulier, nous nous sommes inspirés du principe de **point de variation**. Dans le domaine de la variabilité, cette notion représente simplement un point de modèle concerné par la variation, c'est à dire un endroit où plusieurs options sont possibles. Ces options sont clairement identifiées dans le modèle, et regroupées dans un ensemble appelé les **variants** du point de variation.

2.4.2.2. Point de variation dans les types domaine : le concept *AbstractEntity*

En s'inspirant de cette notion de point de variation, nous avons introduit dans le métamodèle IDEA un concept nommé *AbstractEntity*. Le principe d'une *AbstractEntity* est de matérialiser un point dans une organisation d'entités où plusieurs options sont possibles. Pour une *AbstractEntity* donnée, les options possibles sont restreintes à un ensemble identifié d'entités appelées ses *Variants*. Ces variants peuvent être des entités d'entreprises ou d'autres *AbstractEntity*.

Tous les types de décomposition d'entités d'entreprises pouvant être concernés par la variation, une *AbstractEntity* doit pouvoir être ciblée par les relations *Contains*, *Aggregates*, *Contains* et *Gathers*. Le concept *AbstractEntity* se situe donc au même niveau que *EnterpriseEntity*, c'est-à-dire qu'il dérive du concept *Entity*. Suivant ces principes, la Figure 27 présente le concept *AbstractEntity* dans la vue ENTERPRISE ORGANIZATION du métamodèle IDEA.

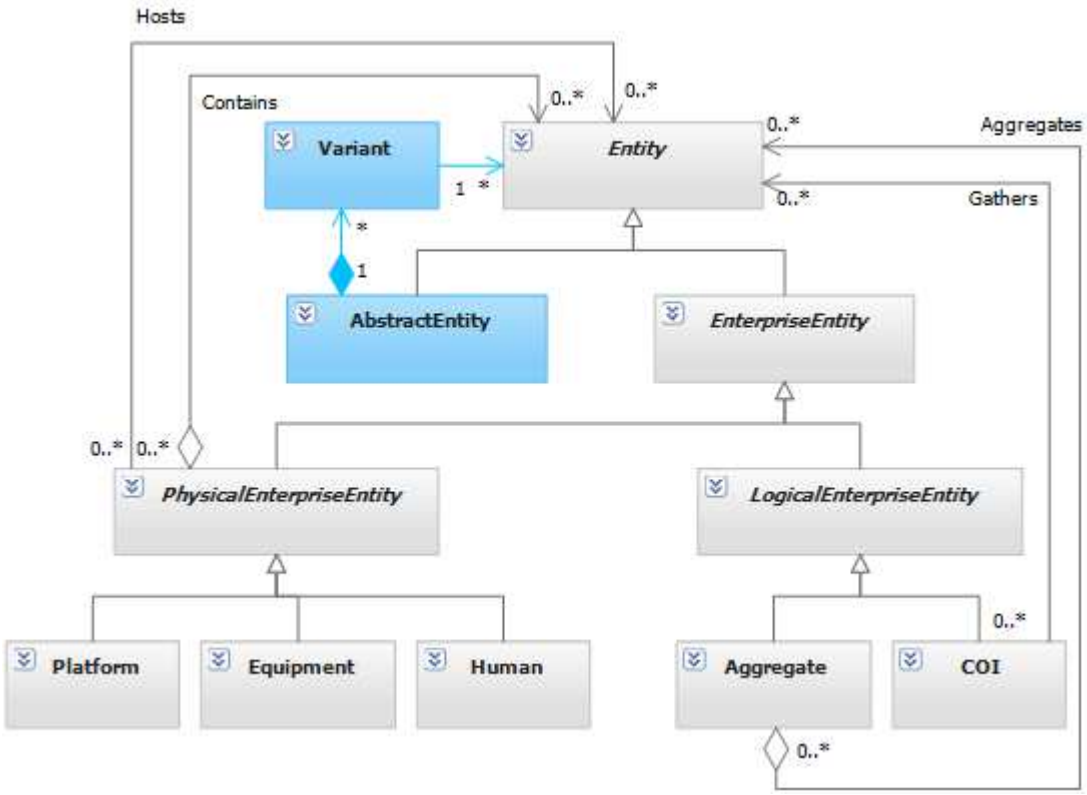


Figure 27 - Le concept *AbstractEntity* dans le métamodèle IDEA

Dans le modèle du convoi, on introduira ainsi au niveau des types domaine une *AbstractEntity* « Moyen de communication », dont les variants sont la « Radio » et la « Radio CDT ». Le type domaine correspondant au blindé sera alors composé du canon, de la mitrailleuse et de cette *AbstractEntity*, comme illustré sur la Figure 28.

Figure 28 – Exemple d'utilisation d'une *AbstractEntity* dans le modèle du convoi

2.4.2.3. Spécification des configurations : le concept *EnterpriseEntityConfiguration*

Dans les approches concernant le domaine d'étude de la variabilité, la représentation de la variation dans un modèle est généralement complétée par un modèle de décision. Ce modèle permet de résoudre la variabilité pour générer un produit déterminé. Par analogie, nous cherchons dans le modèle d'architecture à compléter l'expression des points de variation dans la structure d'un type domaine par la définition d'un modèle de décision permettant d'en dériver une instance domaine déterminée.

Les modèles de décisions sont généralement représentés sous formes de contraintes textuelles. Bien que ce principe soit applicable à notre cas, nous allons plutôt chercher à définir directement dans le modèle la liste exhaustive des configurations possibles pour une entité. En effet, cette approche n'impose pas à l'utilisateur l'apprentissage un langage de contraintes, et permet donc de ne pas complexifier l'effort nécessaire à la modélisation. Cette approche reste cependant plus limitative que l'approche par définition de contraintes[22].

Les entités d'entreprises de niveau type domaine concernées par la définition d'un ensemble de configurations possibles sont celles dont l'ensemble des descendants suivant les relations de décomposition (*Aggregates*, *Contains*, *Gathers* et/ou *Hosts*) contient une ou plusieurs *AbstractEntity*. Ces entités sont alors considérées comme **configurables**. Dans le métamodèle, le concept *EnterpriseEntity* est ainsi doté d'un attribut *IsConfigurable* de type booléen, dont la valeur pour une entité « E » peut être calculée suivant l'algorithme suivant[23] :

```
Pour chaque sous-entité SE composant E
      Si SE est une AbstractEntity
            Alors E.IsConfigurable = true
      Fin si
      Si SE est une EnterpriseEntity et SE.IsConfigurable = true
            Alors E.IsConfigurable = true
      Fin si
      Sinon
            Alors E.IsConfigurable = false
      Fin sinon
Fin pour
```

[22] En particulier, elle impose à toutes les entités d'avoir un ensemble *fini* de considérations possibles.
[23] Dans notre implémentation dans l'environnement DSL Tools, cet attribut est de type *CalculatedProperty* : sa valeur est calculée et mise à jour automatiquement pour chaque élément de modèle et n'est pas éditable par l'utilisateur.

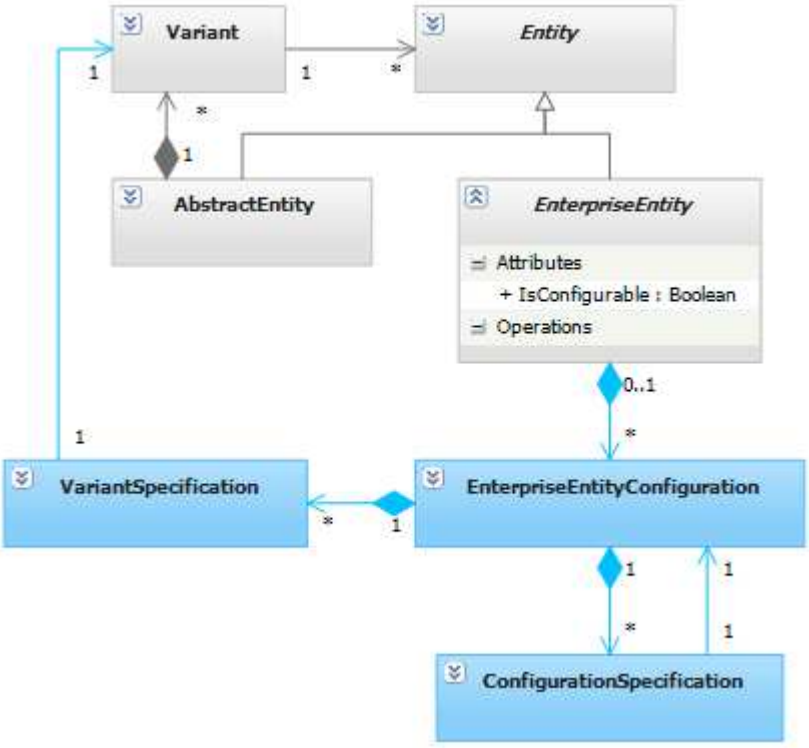


Figure 29 - Le concept *EnterpriseEntityConfiguration* dans le métamodèle IDEA

La définition d'une configuration particulière d'une entité d'entreprise « E » peut s'exprimer en fonction des sous-entités qui la composent :

- Pour chaque sous-entité « SE » de type *AbstractEntity*, la configuration de « E » doit spécifier l'un des variants de « SE ».
  - Par exemple, la configuration « Commandement » du blindé modélisé sur la Figure 30 référence le variant « Radio CDT » de l'*AbstractEntity* « Moyen de communication ».
- Pour chaque sous-entité « SE » de type *EnterpriseEntity* et configurable, la configuration de « E » doit spécifier une configuration de « SE ».
  - Par exemple, on peut définir une configuration « Unité chef de convoi » pour l'*Aggregate* « Unité blindée » contenant le blindé. Cette configuration spécifie alors que le blindé doit être dans sa configuration « Commandement ».

Suivant ce principe, nous avons défini dans le métamodèle un concept *EnterpriseEntityConfiguration* présenté sur la Figure 29. Une *EnterpriseEntityConfiguration* contient deux types de décisions : *VariantRelationship* représente le choix d'un variant pour une sous-entité *AbstractEntity*, et *ConfigurationRelationship* le choix d'une configuration pour une sous-entité *EnterpriseEntity* configurable.

A titre d'exemple, le détail des éléments de modèle composant la configuration « Unité chef de convoi » de l' « Unité blindée » suivant le métamodèle proposé est présenté sur la Figure 30. Pour éviter de surcharger le schéma, les sous-entités communes à toutes les configurations (c.-à-d. les deux armes) ne sont pas représentées.

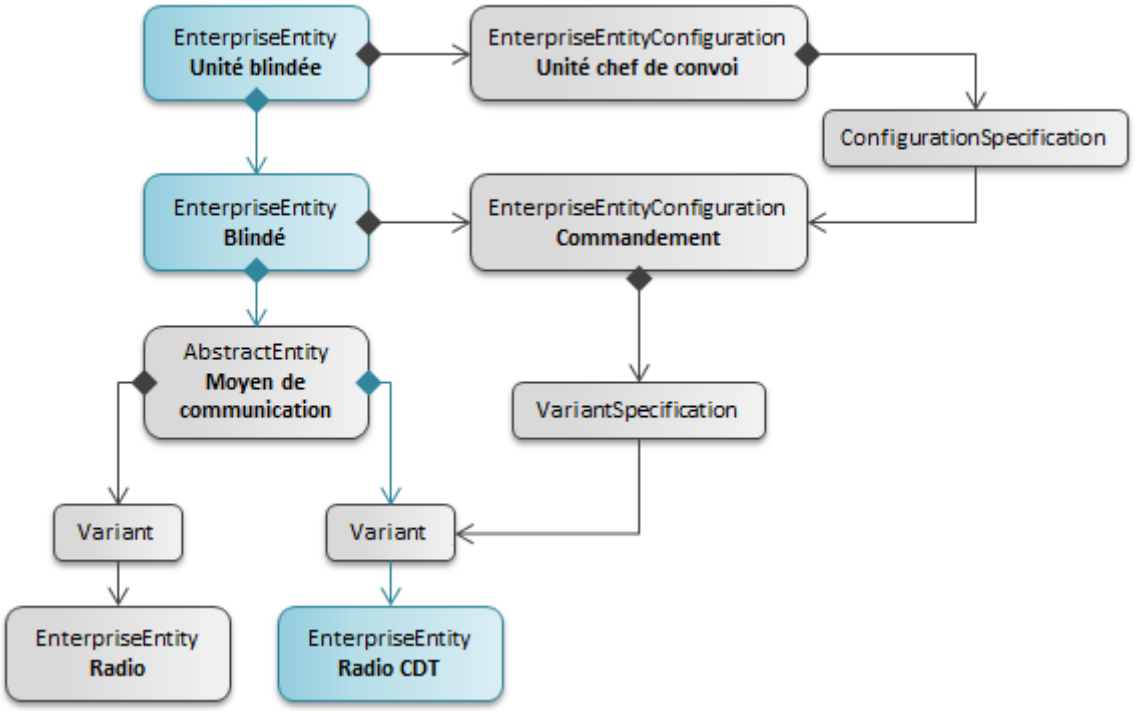


Figure 30 - Représentation détaillée du modèle de la configuration de chef de convoi de l'Unité blindée

Ce métamodèle est complété par un ensemble de règles de cohérence structurelle. En particulier, si l'on considère une *EnterpriseEntity* « E » :

- Si « E » n'est pas configurable, elle ne peut pas contenir d'*EntepriseEntityConfiguration*.
- Pour chaque *VariantSpecification* d'une *EnterpriseEntityConfiguration* de « E », le *Variant* référencé doit appartenir à une *AbstractEntity* faisant partie des sous-entités composant « E ». De même, les *ConfigurationSpecification* doivent référencer une *EnterpriseEntityConfiguration* appartenant à une sous-entité de « E ».

  Chaque *EnterpriseEntityConfiguration* de E doit contenir exactement une *VariantSpecification* pour chaque *AbstractEntity* parmi les sous-entités de « E », et exactement une *Configuration Specification* pour chaque *EnterpriseEntity* configurable parmi les sous-entités de « E ».

Pour instancier une *EnterpriseEntity* configurable, il faut choisir l'une de ses *EnterpriseEntity Configuration*. Les spécifications de cette configuration déterminent alors les sous-entités à instancier. Les *AbstractEntity* ne sont pas présentes au niveau des instances domaines : l'instance du variant sélectionné est directement rattachée à l'entité parente.

Sur la Figure 30 ci-dessus, les *EnterpriseEntity* prises en compte par l'instanciation de la configuration « Unité chef de convoi » de l'« Unité blindée » sont représentées en bleu. Le déploiement correspondant comprendra une instance domaine de l'« Unité blindée » dotée d'un blindé identique à celui présenté dans la partie basse de la Figure 25.

2.4.3. Configurations des rôles / collaborations

2.4.3.1. Potentiel d'interaction d'un rôle : le concept *RoleInterface*

Pour la collaboration qui le contient, un rôle est vu en « boîte noire », c'est-à-dire uniquement par sa description en termes de produits (ou d'évènements) attendus et envoyés, et de services consommés et fournis. Cette vue représente l'**interface** du rôle, c'est à dire uniquement son **potentiel d'interaction** au sein de la collaboration sans présumer de la manière dont il est conçu. Dans le langage IDEA, ce potentiel d'interaction est représenté par un ensemble d'éléments *Endpoint*. A l'opposé, l'ensemble des sous-rôles et des interactions entre ces sous-rôles constitue une vue en « boîte blanche » d'un rôle. Cette vue représente l'**implémentation** du rôle, c'est-à-dire la manière dont il s'organise **en interne** pour être capable interagir avec d'autres rôles dans une collaboration.

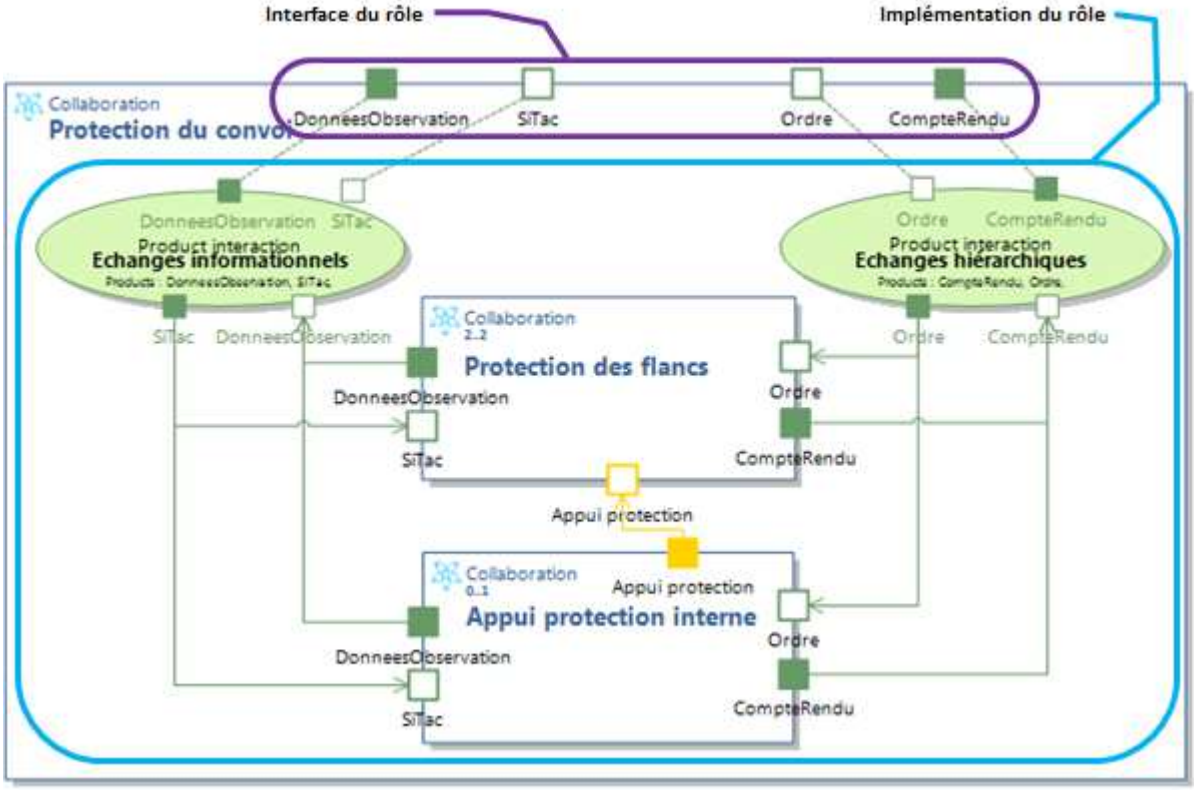


Figure 31 - Interface vs implémentation d'un rôle

Ces deux vues d'un même rôle sont illustrées sur l'exemple du rôle « Protection du convoi » sur la Figure 31. Son interface est composée des *Endpoints* d'envoi de données d'observation et de comptes-rendus, et de réception de la situation tactique (SiTac) et d'ordres. Son implémentation comporte les deux sous-rôles « Appui protection » et « Protection des flancs », ainsi que les interactions « Echanges informationnels » et « Echanges hiérarchiques » qui font le lien avec l'interface.

Dans le métamodèle, l'interface et l'implémentation d'un rôle ont été décorrélées par l'introduction du concept *RoleInterface* présenté sur la Figure 32 (évolution du schéma de la Figure

11). Chaque *Role* contient une *RoleInterface*, composée de l'ensemble des *Endpoint* dont il dispose et qui lui permettent d'interagir au sein d'une collaboration.

Les liens entre l'interface et l'implémentation d'un rôle sont modélisés par des relations *Delegates* établies entre deux *Endpoints*. L'*Endpoint* source de la relation doit appartenir à l'interface du rôle, et la cible soit à l'interface d'un de ses sous-rôles, soit à une interaction de produits (ou d'évènements) comme dans le cas du rôle « Protection du convoi ». Dans le DSL IDEA, cette contrainte est implémentée sous forme de règle de cohérence. Elle impose de plus aux deux *Endpoints* d'être équivalents, c'est-à-dire de même type (ex. *ProvidedServiceEndpoint*) et de même caractéristique (ex. référencer le même service).

Sur la Figure 31, les relations *Delegates* sont représentées par les liens en pointillés.

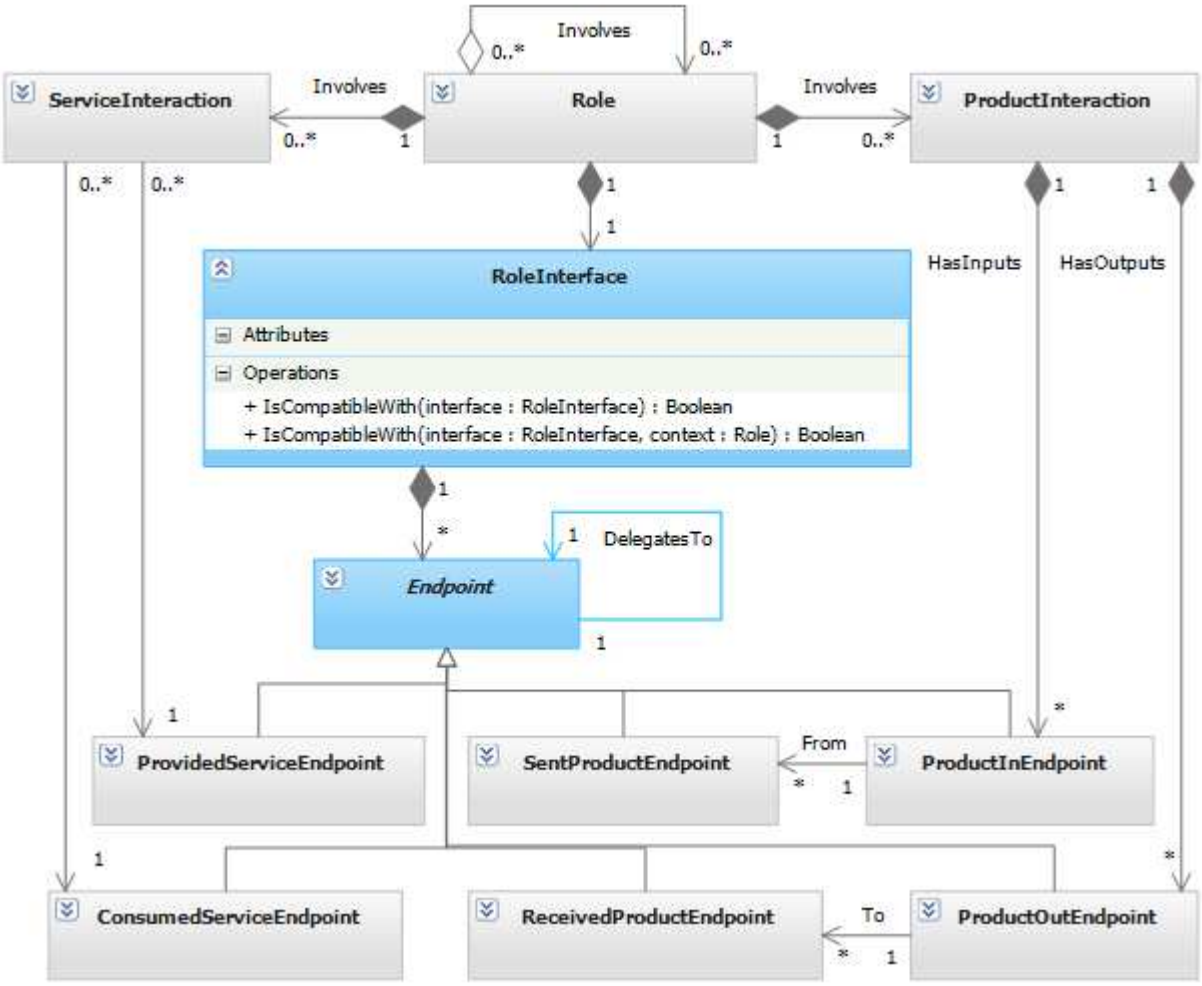


Figure 32 - Le concept *RoleInterface* dans le métamodèle IDEA

Le concept *RoleInterface* est de plus doté d'une opération `IsCompatibleWith()` prenant en paramètre une autre interface et retournant un booléen. Une interface est considérée compatible d'une autre si elle contient au moins un *Endpoint* équivalent à chacun des *Endpoints* de l'autre. La compatibilité des interfaces n'est pas symétrique : sur l'exemple illustré sur la Figure 33, l'interface du rôle « Appui protection externe » est compatible de celle de « Appui protection interne », mais l'inverse est faux. En effet, il manque à l'interface « Appui protection interne » un *Endpoint* de réception de confirmation de l'ordre.

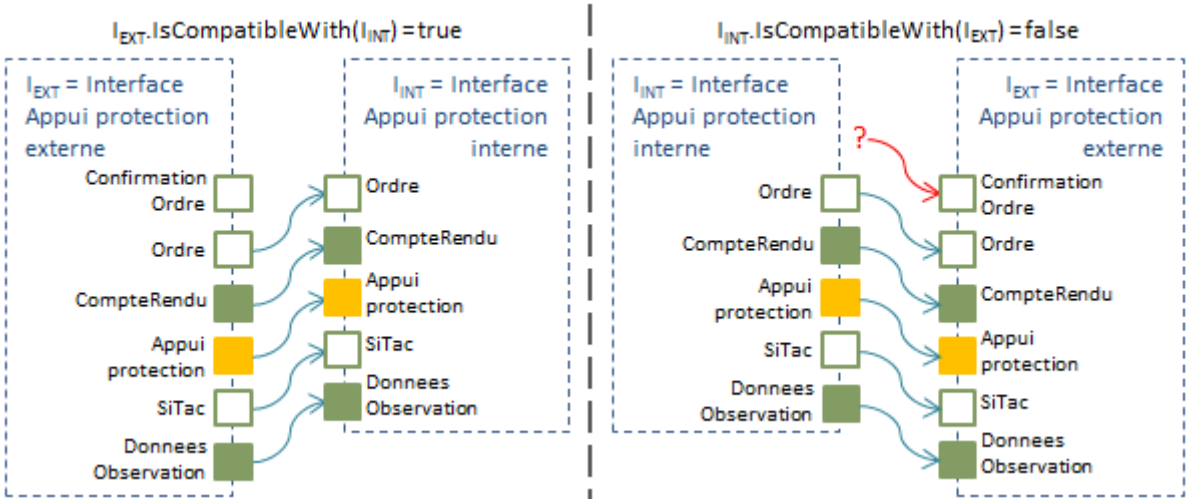


Figure 33 - Principe de compatibilité des *Interfaces*

2.4.3.2. Interface requise par une collaboration et interchangeabilité des rôles

Sur le principe, la compatibilité des interfaces permet aux rôles qui les portent d'être **interchangeables**. Du point de vue d'une collaboration, cela signifie qu'elle pourrait remplacer n'importe lequel de ses rôles par un autre, pourvu qu'il soit doté d'une interface compatible et donc capable d'assumer toutes les interactions attendues. Par exemple, la collaboration « Protection du convoi » de la Figure 31 pourrait remplacer son rôle d' « Appui protection » par n'importe quel autre rôle disposant d'une interface contenant au minimum quatre *Endpoints* d'envoi des données d'observation, d'envoi de comptes-rendus, de réception de la SiTac et de réception d'ordres, et un *Endpoint* de fourniture du service « Appui protection ».

Sur la base de ces constatations, il apparait légitime de reconsidérer la pertinence de l'établissement au niveau métamodèle de la relation *Involves* entre rôles. Comme on peut l'observer sur la Figure 32, celle-ci est en effet établie entre un rôle source et un rôle cible. Afin de mieux mettre en exergue l'interchangeabilité des rôles et le fait qu'une collaboration ne se préoccupe que de l'interface de ses rôles, elle aurait pu au contraire cibler le concept *RoleInterface*. Dans ce cas, le choix et l'acte d'introduire un rôle compatible dans une collaboration pour remplir une interface requise serait entièrement à la charge de l'utilisateur. Dans le cas d'une organisation complexe de rôles, cette charge pourrait se révéler importante. Par exemple pour la collaboration « Protection du convoi », l'utilisateur devrait en premier lieu définir les trois interfaces correspondant à l'appui protection et aux protections des flancs, indépendamment de ces interfaces définir les trois rôles « Appui protection », « Protection flanc gauche » et « Protection flanc droit », puis revenir au niveau de la collaboration de protection pour associer individuellement chacun de ces rôles aux interfaces. L'ensemble de ces actions requiert plus d'effort que de définir directement les rôles dans la collaboration.

Dans le cadre de notre domaine d'application, dans la majeure partie des cas la modélisation des structures de rôles concerne une décomposition précise et identifiée d'une collaboration en rôles. Faire intervenir de la variabilité et chercher à remplacer un rôle par un autre est ainsi une action certes nécessaire, mais devant rester optionnelle car ne représentant pas la majorité des cas. Par conséquent, la réalisation d'un modèle où l'arborescence des rôles est déterminée doit être celle qui nécessite le moins d'effort à l'utilisateur, et donc l'approche prévue par défaut par le langage. Plutôt que d'obliger dans tous les cas la modélisation indépendante des interfaces requises par les

collaborations et des rôles qui en sont compatibles, nous avons préféré l'approche de modélisation directe des rôles dans la collaboration.

Par rapport au principe d'instanciation, la définition d'un rôle dans une collaboration au niveau des types domaines représente une **spécification par défaut** du rôle attendu. Nous considérons alors l'interface minimale requise par une collaboration pour un rôle comme déduite de l'ensemble des **interactions** dans lesquelles la collaboration fait intervenir ce dernier. Il s'agit alors des *Endpoints* du rôle impliqués dans une interaction dans le cadre de la collaboration, et donc potentiellement un sous-ensemble de l'interface propre du rôle plutôt que la totalité. Par exemple, considérons une collaboration « R_C » dans laquelle a été défini un rôle « R_A » porteur de quatre *Endpoints*, mais ne faisant intervenir ce rôle que dans deux interactions. Comme illustré sur la Figure 34, l'interface requise par « R_C » pour « R_A » est un sous-ensemble de l'interface de ce dernier, limité aux deux *Endpoints* impliqués dans les interactions.

Concrètement, nous n'avons pas intégré cette notion d'interface requise au niveau du métamodèle IDEA puisqu'elle peut être déduite par un algorithme très simple. Nous l'avons implémentée dans une méthode surchargeant la méthode `IsCompatibleWith()` présentée en 2.4.3.1, prenant un paramètre supplémentaire de type *Collaboration* qui représente la collaboration définissant les interactions à prendre en compte. Dans le cas de l'exemple présenté par la Figure 34, si l'on note « I_A » l'interface du rôle « R_A » et « R_B » l'interface du rôle « R_B », on aura donc I_B`.IsCompatibleWith(`I_A`) = false`, mais I_B`.IsCompatibleWith(`I_A`, `R_C`) = true` : « I_B » ne contient pas d'équivalent à l'intégralité des *Endpoints* de « I_A », mais contient un équivalent pour chaque *Endpoint* de « I_A » impliqué dans une interaction dans le contexte de « R_C ».

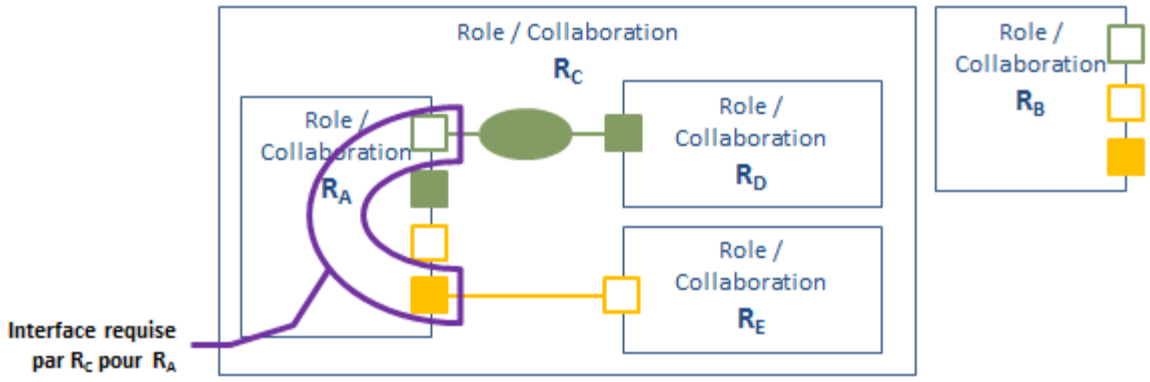


Figure 34 - Interface requise par une collaboration pour un rôle

A l'instanciation, les rôles d'une collaboration sont instanciés avec elle et le déploiement ainsi obtenu est complet et fonctionnel sans requérir d'action de la part de l'utilisateur. Ce dernier peut cependant exploiter l'interchangeabilité des rôles pour remplacer un rôle instance domaine par un autre, sous contrainte d'une règle du langage imposant la compatibilité de l'interface de ce nouveau rôle avec l'interface requise par la collaboration. Par exemple, après instanciation de la collaboration « R_C », l'utilisateur peut remplacer l'instance du rôle « R_A » par n'importe quel rôle instance porteur de deux *Endpoints* compatibles, comme par exemple « R_B ». On dira alors que le nouveau rôle instance participe à la collaboration **au titre de** l'ancien.

2.4.3.3. Participation d'un rôle à plusieurs collaborations

Comme illustré sur la Figure 34, il peut arriver qu'un rôle de niveau instance domaine vienne participer à une collaboration sans pour autant impliquer tous ses *Endpoints* dans des interactions. Dans de telles situations, il est possible que le rôle ne reçoive pas l'ensemble des entrées nécessaires à son bon fonctionnement. Par exemple, considérons un rôle « Appui protection externe » tel que présenté sur la Figure 33, participant à la « Protection du convoi » au titre de « Appui protection interne ». Son interface comporte un *Endpoint* de réception de confirmation d'ordre, qui n'est pas impliqué dans une relation de service dans le cadre de « Protection du convoi ». Or la réception de ce produit est, selon la doctrine, indispensable à ce rôle pour pouvoir fournir le service « Appui protection ».

Obliger une collaboration à impliquer tous les *Endpoints* de ses rôles dans des interactions reviendrait à invalider le principe d'interface requise par une collaboration pour un rôle tel qu'il a été défini précédemment, et priver ainsi le modèle de la flexibilité qu'il apporte. De plus, le fait que tous les *Endpoints* ne soient pas connectés peut également ne pas constituer un obstacle au bon fonctionnement du rôle, voire même être nécessaire dans le cadre de l'étude de l'architecture (par exemple pour modéliser et évaluer des modes dégradés).

Pour permettre à un rôle de recevoir l'ensemble de ses entrées tout en étant impliqué dans une collaboration qui ne définisse pas comment elles lui sont fournies, il est nécessaire d'autoriser un rôle de niveau instance domaine à prendre part à plusieurs collaborations. Cette participation à plusieurs collaborations ne doit cependant pas entraîner de confusion possible à l'exécution. En particulier, un service ne peut être consommé que depuis un seul fournisseur (afin d'éviter de devoir faire un choix par défaut quant à l'appel). Par conséquent, un rôle instance domaine ne peut participer à plusieurs collaborations que si les interfaces requises par les collaborations ne contiennent pas d'*Endpoints* équivalents en ce qui concerne la consommation de service. Ce principe est illustré sur la Figure 35. Dans le DSL IDEA, ces restrictions ont été implémentées dans des règles de cohérence limitant les choix de l'utilisateur aux rôles conformes.

Deux collaborations partageant un rôle établissent par ce biais un lien de dépendance. En particulier, une interaction dans l'une peut être conditionnée par la mise en œuvre d'une interaction dans l'autre. C'est le cas de la collaboration « Protection du convoi » dans sa configuration d'appui externe : la fourniture du service d'appui protection ne peut pas avoir lieu si, au préalable, le rôle « Appui protection externe » n'a pas reçu confirmation de l'ordre, laquelle doit lui être fournie dans le cadre d'une collaboration différente. Lors de la simulation, deux collaborations partageant un rôle peuvent ainsi influer l'une sur l'autre d'un point de vue temporel, voire se synchroniser. Dans le cas de la collaboration « Protection du convoi », les deux rôles de protection resteront bloqués en attente de l'appui aussi longtemps que la confirmation de l'ordre ne sera pas parvenue à l'« Appui protection externe ». A noter qu'il ne s'agit pas là d'un impact restrictif sur le modèle : cette situation est représentative de la réalité et permet d'identifier et d'anticiper des cas de blocage de l'entreprise considérée.

2.4.3.4. Par rapport aux acteurs…

Faire participer un rôle à plusieurs collaborations est d'un esprit différent de faire jouer des rôles différents à un même acteur. En effet, le premier cas induit une dépendance entre les collaborations,

alors que les différents rôles joués par un même acteur le sont de manière indépendante. De même, le principe d'interchangeabilité des rôles est complémentaire de la possibilité de substituer un acteur à un autre pour jouer un même rôle, à la fois du point de vue du sens du modèle que de son exécution.

Faire participer un rôle dans une collaboration au titre d'un autre rôle signifie modifier la manière dont les responsabilités allouées par la collaboration à ce rôle (c.-à-d. interactions liées à l'interface requise) sont mises en œuvre. Une telle modification aura généralement un impact à la simulation sur le chemin d'exécution, dans la mesure où des processus différents seront déroulés, par des acteurs différents et dans le cadre d'interactions différentes.

En revanche, remplacer l'acteur qui joue un rôle (*via* la relation *Plays*) par un autre ne modifie pas la manière, mais les moyens par lesquels les responsabilités allouées à ce rôle sont mises en œuvre. Cette modification peut par exemple avoir un impact sur la disponibilité du rôle en alourdissant la charge de l'acteur qui le joue, mais pas sur le chemin d'exécution.

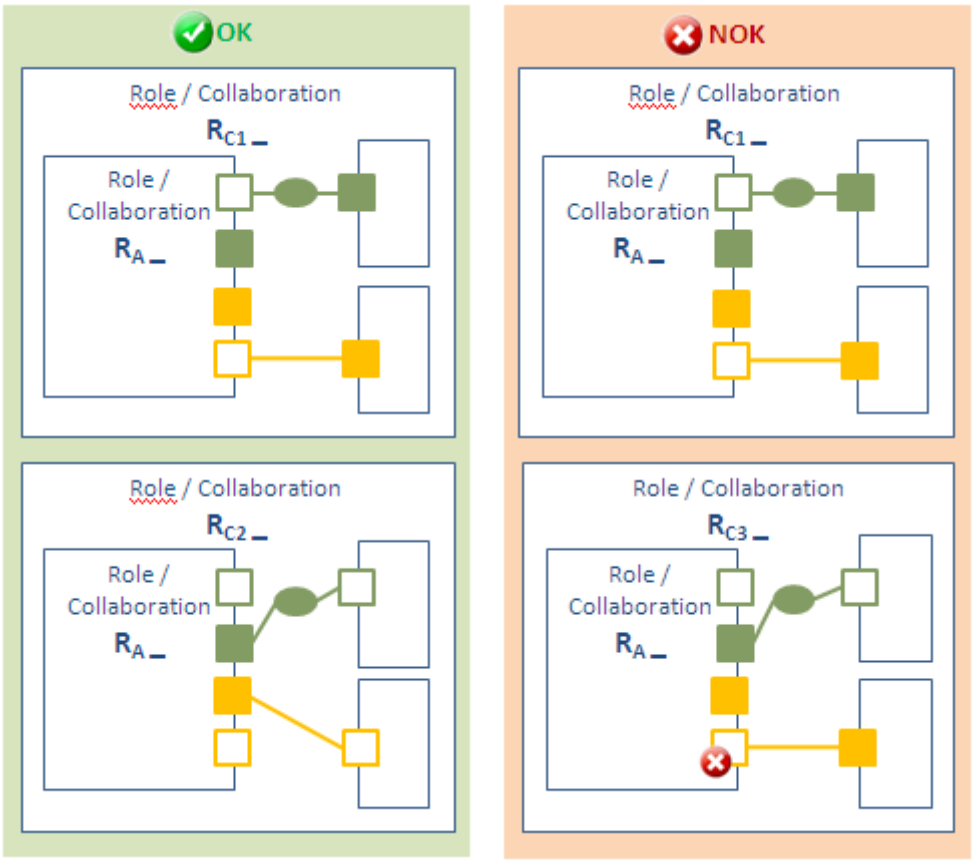


Figure 35 - Exemple de restrictions sur la participation d'un rôle à plusieurs collaborations

Chapitre 3. Application et retours d'expérience

3.1. Implémentation et outillage

3.1.1. Mise en œuvre du langage et des contributions de la thèse

3.1.1.1. Contexte technologique

Pour le DSL IDEA, l'environnement DSL Tools de Microsoft a été privilégié par rapport aux environnements EMF : dans une optique de simulation du modèle, il est en effet apparu essentiel de disposer d'un environnement permettant de personnaliser aisément le code associé aux classes générées à partir du métamodèle. Par exemple, pour pouvoir estimer le contexte d'un processus en cours d'exécution, il est indispensable de définir des méthodes de parcours du modèle pouvant être exploitées par l'environnement de simulation. L'environnement DSL Tools permet de tirer parti du mot-clé .NET « partial », supportant la définition d'une même classe sur plusieurs fichiers, et donc une séparation propre entre le code généré à partir du métamodèle et le code personnalisé.

De plus, la version la plus récente de DSL Tools à la date des travaux permet d'utiliser Windows Presentation Foundation (WPF) pour la représentation graphique des modèles. Ce formalisme générique offre un éventail de possibilités plus large que le Graphical Modeling Framework (GMF) utilisé par EMF et dédié à la modélisation. Les modèles créés sur la base du langage IDEA pouvant être utilisés comme supports de communication entre les intervenants de l'architecture, leur représentation graphique présente une grande importance bien qu'elle ne soit pas adressée en détail dans le cadre de cette thèse.

3.1.1.2. Architecture du DSL IDEA dans DSL Tools

L'architecture du langage telle que présentée en 2.2.1.1 se traduit dans le formalisme DSL Tools de la manière suivante :

- Le **métamodèle** est défini par le composant principal des DSL Tools, le **DslDefinition**. Il est construit de manière graphique, et permet ensuite de générer du code définissant la structure technique du langage. A la génération, une **classe C#** est créée pour chaque **concept** et chaque **relation** du métamodèle.
- Les **règles de cohérence** se traduisent dans DSL Tools par des **Consistency Rules**, c'est à dire des classes C# dotées d'un attribut particulier.
  Les règles de cohérence sont principalement caractérisées par les éléments suivants :
  - Le ou les concept(s) ou relation(s) du métamodèle sur lequel (lesquels) porte la règle,
  - Un ou plusieurs événement(s) déclencheur(s) de la règle (ex. création d'un nouvel élément de modèle issu d'un concept concerné par la règle),
  - Pour chaque événement déclencheur, une fonction C# définissant le comportement de la règle (ex. création automatique d'un élément lié).
- Les règles de validation se traduisent dans DSL Tools par des **Validation Rules**, c'est à dire des classes C# dotées d'un attribut particulier.
  Les règles de validation sont principalement caractérisées par les éléments suivants :
  - Le ou les concept(s) ou relation(s) du métamodèle sur lequel (lesquels) porte la règle,

- Une fonction vérifiant un certain nombre de caractéristiques de structure de l'ensemble des éléments de modèle issus d'un concept concerné par la règle,
- Un ou plusieurs messages à afficher à l'utilisateur selon le résultat des vérifications effectuées.

Nous avons ajouté aux classes générées à partir du métamodèle (concepts et relations) de deux types de méthodes auxiliaires :

- Les méthodes métier, décrivant des **fonctionnalités métier** comme par exemple les méthodes d'instanciation `Instantiate()` présentées en 2.3.3.5. Ces méthodes sont **appelées par l'environnement de modélisation** au cours de la définition d'un modèle, comme les règles de cohérence et de validation.
- Des méthodes d'**exploration du modèle**, constituant une API permettant à un programme tiers de **parcourir un modèle sans connaître le détail de sa structure**. Par exemple, une méthode sur la classe du concept *EnterpriseEntity* retourne, pour une entité de niveau instance donnée, l'entité de niveau instance lui fournissant un service donné dans le cadre d'une collaboration instance donnée.

Du point de vue du projet logiciel, ces méthodes sont rédigées dans des fichiers différents de ceux contenant le code généré à partir du métamodèle.

Les classes générées à partir du métamodèle, leurs méthodes auxiliaires, les règles de cohérence et les règles de validation constituent une DLL autonome (DSLDefinition.dll). Cette DLL est exploitée par le moteur d'exécution en charge de la simulation des modèles.

Le moteur d'exécution, dont le contenu technique sort du périmètre de cette thèse, se base sur le formalisme Microsoft Workflow Foundation (WF) pour exécuter les processus définis dans un modèle : l'exécution fait intervenir, de manière transparente pour l'utilisateur, une traduction des éléments de modèle de type *Process* dans le format natif (WF). Il exploite les méthodes d'exploration fournies par la DLL DSLDefinition.dll pour mettre en œuvre les interactions (services, produits…) et retrouver toutes les autres informations qui peuvent lui être nécessaires.

La Figure 36 ci-après présente un récapitulatif de cette architecture, en lien avec la Figure 6 présentée en 2.2.1.1.

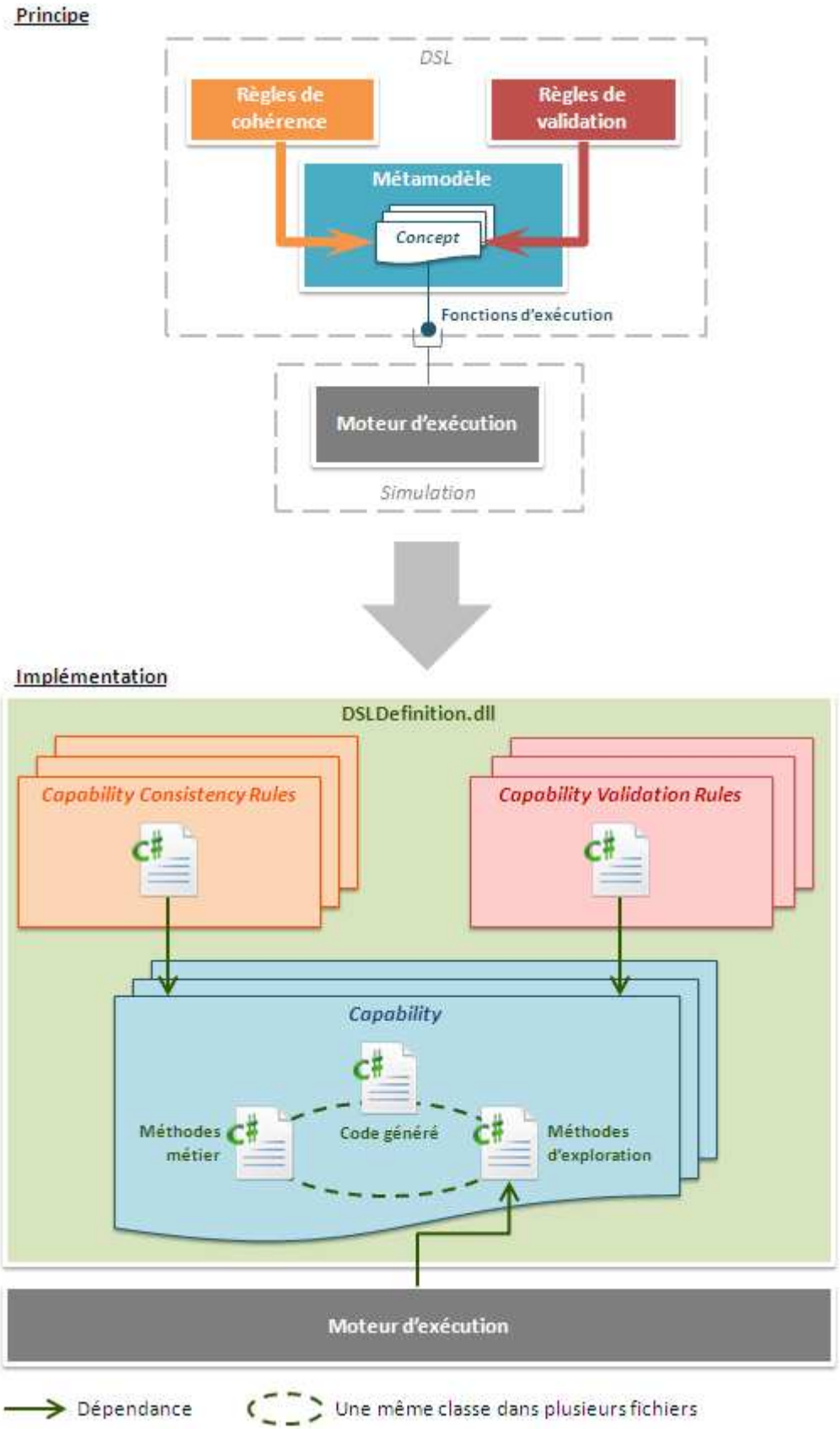


Figure 36 - Implémentation de l'architecture du langage dans MS DSL Tools

3.1.1.3. Présentation de la suite IDEA Studio

Au cours de la démarche IDEA présentée en 1.3.2.2 sont mis en œuvre un certain nombre d'outils, dont les grandes familles sont représentées sur la Figure 37. En support aux aspects prototypage rapide (boucles I-D et D-E principalement) a été développée une suite logicielle dédiée, **IDEA Studio**, basée sur le langage de description d'architecture présenté dans cette thèse. L'objectif de la suite logicielle est de permettre de définir, valider et évaluer des architectures d'entreprises et de Systèmes de Systèmes par le biais d'un modèle simulable.

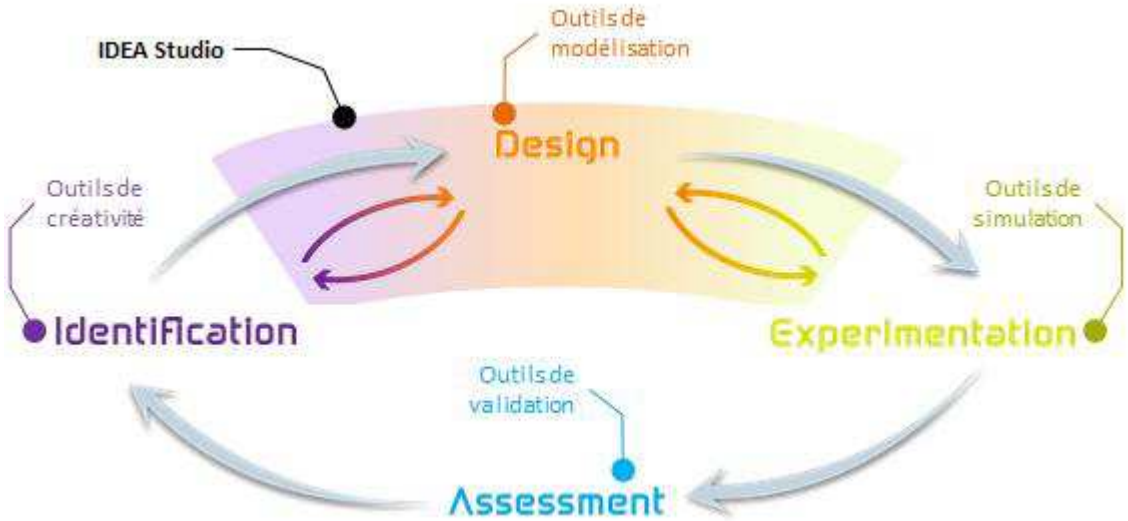


Figure 37 - Outillage de la démarche IDEA

IDEA Studio est composée de deux outils : IDEA Designer pour la modélisation, et IDEA Performer pour la simulation. Les deux outils exploitent un modèle unique, persistant sous la forme d'un fichier xml. Ce principe permet d'éviter toute transformation de modèle, conformément au besoin identifié en 1.2.1.3 pour le support à une approche par prototypage rapide.

A la date de rédaction de ce document, les deux outils sont d'un bon niveau de maturité et font l'objet d'une utilisation dans le cadre d'affaires industrielles, dont un modèle sera présenté en 3.2.

IDEA Designer est l'outil de modélisation, permettant de représenter une architecture selon le DSL IDEA. Il permet la visualisation et l'édition du modèle par le biais d'un ensemble de diagrammes, présentés plus en détail en Annexe 6.

IDEA Designer est constitué de la DLL contenant le DSL IDEA implémenté dans DSL Tools, telle que présenté dans la partie précédente, et de l'IHM de modélisation permettant de créer les modèles. La majeure partie du code source de l'outil, environ 85% en termes de nombre de lignes, est constitué par le code généré à partir du métamodèle.

Comme pour tous les modeleurs produits par DSL Tools, l'IHM de modélisation d'IDEA Designer est basée le Shell Visual Studio dans sa version 2010 ou ultérieure. Le Shell est une exécution « vide » de Visual Studio, dans laquelle est intégré le modeleur. L'allure générale de l'IHM est ainsi similaire à celle de Visual Studio. Les diagrammes de modélisation (cf. Annexe 6) exploitent soit directement les fonctionnalités graphiques offertes par DSL Tools (Windows Form), soit le formalisme Windows Presentation Foundation (WPF).

Une capture d'écran représentant l'édition d'un modèle dans cet outil est présentée en Annexe 4.

IDEA Performer est l'outil de simulation, basé sur le moteur d'exécution permettant de simuler les processus métier dans le contexte de l'architecture déployée. L'exécution du modèle peut être visualisé selon différentes perspectives, comme les interactions représentées entre les entités déployées sous forme de pions sur une carte, une vue chronologique de l'exécution de l'ensemble des processus, ou encore une visualisation individuelle des processus. La simulation des processus peut se faire soit en pas à pas, soit en continu. Dans tous les cas, l'utilisateur peut intervenir pendant l'exécution comme par exemple déplacer des pions sur la carte, forcer des décisions dans les processus et injecter des événements.

IDEA Performer se base sur le moteur d'exécution de Windows Workflow Foundation (WF), et présente une IHM développée intégralement avec la technologie WPF. Pour parcourir, interroger, voire modifier le modèle, il fait appel à une API spécifique définie au niveau du DSL dans le projet d'IDEA Designer.

Une capture d'écran représentant l'exécution d'un modèle dans cet outil est présentée en Annexe 5.

3.1.1.4. Intégration des travaux de thèse

Les travaux réalisés au cours de cette thèse pour adresser les problématiques d'adaptabilité de l'entreprise n'ont pas tous été intégrés au même stade dans l'outil IDEA Designer : certains sont intégrés à la version en cours à la date de rédaction de cette thèse (version dite « actuelle » dans la suite de cette partie), d'autres sont implémentés sous forme de prototype. Les travaux adéquats ont été par ailleurs réalisés par l'équipe de développement au niveau d'IDEA Performer pour que toutes les contributions intégrées à soient prises en compte lors de l'exécution du modèle.

Le choix de l'implémentation dans la version actuelle de l'outil de chacune des contributions a été évalué sur la base des besoins des utilisateurs, l'outil étant exploité dans des affaires industrielles. L'accent a été ainsi principalement mis sur les travaux conditionnant la structure de base d'un modèle, à savoir l'organisation du métamodèle et le principe d'instanciation domaine.

Les aspects de configuration des entités, estimés moins prioritaires que la configuration des rôles, seront intégrés à la version suivante.

Le Tableau 4 ci-dessous présente une vue synthétique du statut de l'implémentation des contributions telles que développées au Chapitre 2 :

Contribution	Statut
2.2 Organisation du langage de description d'architecture	
Mise en œuvre des vues dans le métamodèle IDEA	Intégré
2.3 Types et instances domaines	
Types et instances domaine dans le métamodèle	Intégré
Algorithmes d'instanciation, règles de maintien en cohérence	Intégré
2.4 Modélisation de configurations dans l'entreprise	
Configuration des entités : *AbstractEntity* et *EntityConfiguration*	Prototype
Configuration des rôles : *RoleInterface*, interchangeabilité des rôles, rôles partagés	Intégré

Tableau 4 - Intégration des travaux de thèse dans IDEA Studio

Les travaux réalisés dans le cadre de cette thèse ont été orientés par l'ensemble d'objectifs introduits en 1.3.2.1. La contribution à ces objectifs étant non quantifiable, nous allons principalement l'évaluer par le biais des **retours utilisateurs**, recueillis aussi bien auprès d'utilisateurs-testeurs que d'utilisateurs dans le cadre d'affaires. Au-delà de l'évaluation des objectifs et du strict cadre de l'adaptabilité ayant guidé cette thèse, dans l'ensemble de cette partie nous nous appuierons également sur ces retours utilisateurs et notre expérience suite à la mise en œuvre du langage pour identifier des **points forts et des recommandations générales quant à la conception d'un DSL de description d'architecture d'entreprise visant à produire des modèles simulables**.

3.1.2. Retours d'expérience sur l'implémentation du langage

3.1.2.1. Retours sur les objectifs

La conception du DSL IDEA a nécessité une attention particulière à l'expressivité du langage pour l'utilisateur. Cette préoccupation, traduite par les objectifs d'**expressivité métier** et de **généricité** ([EL1] et [EL2]), a été en particulier adressée par le biais de l'utilisation de concepts adaptés. Le langage IDEA a été ainsi conçu autour d'un ensemble de concepts-clés, choisis pour leur expressivité vis-à-vis de la terminologie en usage pour l'architecture d'entreprise sans toutefois être spécifiques à un domaine d'application. Le choix de ces concepts-clés s'est révélé pertinent pour les utilisateurs, dans la mesure où il leur a permis de transcrire leur métier dans le langage de modélisation de manière relativement naturelle.

Certains des concepts reprennent cependant des termes utilisés dans des communautés différentes, et avec un sens différent. C'est notamment le cas du concept de « Service ». Pour les ingénieurs, en particulier issus du domaine des technologies de l'information, ce terme est étroitement lié aux principes de l'architecture orientée services (SOA) [41] et aux web services. Dans le métamodèle, ils ont donc par exemple regretté de ne pas retrouver la notion de « contrat ». Pour les experts opérationnels en revanche, le concept de service a un sens plus large et moins formalisé, et cette partie du métamodèle n'a pas posé de problème particulier vis-à-vis de leur vision du terme.

Pour définir dans le métamodèle IDEA ces concepts partagés entre plusieurs communautés malgré le manque de vision fédératrice (comme pourrait par exemple le proposer un métamodèle NAF formel), nous avons dû faire des choix pour adopter des compromis entre les différentes définitions (par exemple, ne pas inclure de concept de « contrat » de service). Pour valider la pertinence des choix effectués pour la définition du métamodèle, deux études ont été menées hors du périmètre de cette thèse, l'une concernant la traduction d'un modèle IDEA vers NAF et l'autre cherchant à reproduire un modèle de Bayne. A titre d'information, les résultats de ces deux expérimentations sont présentés en Annexe 7 et Annexe 8.

En support à l'approche de prototypage rapide dans laquelle se positionne le langage IDEA, l'objectif de **modélisation assistée** ([EL5]) s'est traduit par la mise en place d'un ensemble de règles de cohérence et de validation. C'est alors l'action combinée de toutes ces règles qui permet d'assurer la **cohérence** et l'**exécutabilité** du modèle ([EM1] et [EM2]).

3.1.2.2. Analyse des règles mises en œuvre

En particulier, les règles visant au maintien en cohérence des instances domaines par rapport à leur type, exposées en 2.3.4, sont très représentatives des enjeux et problématiques rencontrés pour la mise en œuvre de l'ensemble des règles du langage. Au-delà de ces règles définies dans le cadre de la thèse, les règles de cohérence et de validation qui ont été nécessaires pour mettre en œuvre la modélisation assistée peuvent se classer selon deux axes : leur objectif et leur mode d'action. Il importe de noter que cette classification n'est pas propre au domaine des entreprises et des Systèmes de Systèmes, et peut être prise en compte pour assurer un objectif similaire dans tout autre DSL que le langage IDEA.

- Axe des objectifs :
  - **Règles de complétude** – Ces règles visent à assurer que l'ensemble des éléments nécessaires à la simulation et à la conformité du modèle au langage ont été modélisés.
  - **Règles de structure** – Ces règles visent à empêcher l'établissement de patterns incohérents vis-à-vis des contraintes d'architecture.
- Axe des modes d'action :
  - **Règles passives** – Ces règles effectuent une vérification des modifications effectuées, afin d'interdire certaines actions ou de remonter des informations à l'utilisateur.
  - **Règles actives** – Ces règles supervisent les actions de l'utilisateur en effectuant des actions automatiques sur le modèle.
    Pour un DSL créé dans l'environnement DSL Tools, toutes les règles actives sont des règles de cohérence étant donné que les règles de validation ne peuvent pas modifier le modèle.

Des exemples illustrant ces différentes catégories de règles dans le DSL IDEA sont présentés dans le Tableau 4 :

	Règles de complétude	**Règles de structure**
Règles passives	Une règle de validation vérifie que tous les rôles de niveau instance domaine sont affectés à une instance d'entité, et indiquant à l'utilisateur lesquels ne le sont pas.	Une règle de cohérence interdit l'établissement d'une relation de service entre deux *Endpoints* qui ne concerneraient pas le même service.
Règles actives	A la création d'une *Capability*, une règle de cohérence crée un élément *Process* (par défaut, vide) et le lui associe.	Un ensemble de règles de cohérence assurent la propagation des modifications effectuées sur une *Collaboration* de niveau type domaine vers ses instances.

Tableau 5 - Classification des règles de cohérence et de validation : exemples

L'objectif de **maîtrise de la complexité** ([EL4]) est principalement adressé dans le DSL IDEA par le principe de types domaines et instances domaine présenté en 2.3.2. L'outillage, en particulier par le biais des IHM de présentation du modèle, joue également un rôle important pour cet objectif.

L'objectif d'**expressivité architecturale** ([EL5]) se traduit dans le DSL IDEA par l'ensemble des mécanismes présentés dans le Chapitre 3. Les relations entre concepts appartenant à des vues différentes (ex. la relation *Plays* entre un rôle et l'acteur qui le joue), les différentes instances domaines correspondant potentiellement à un même type et les configurations constituent autant de points d'articulation de l'architecture et du modèle : ce sont des endroits où tous deux peut être

aisément modifiés, avec un impact mineur sur le reste des constituants. C'est par le biais de cet impact réduit au minimum que peut se faire la **modification dynamique** ([EM3]) d'un modèle au niveau des points d'articulation.

Les retours sur ces trois objectifs seront présentés dans le cadre de l'application du langage dans une affaire réelle en 3.2.

3.1.2.3. Analyse des points clés pour une démarche de conception d'un DSL outillé

Pour valider au plus tôt le langage et les contributions présentés dans cette thèse et s'assurer à la fois de leur utilisabilité et de leur acceptation par les utilisateurs, il nous est apparu nécessaire de concevoir le langage en adoptant un développement évolutif. Ce n'est que par une telle approche, prenant en compte les remarques des utilisateurs vis-à-vis des choix de conception, qu'a pu être défini avec succès un langage de modélisation outillé s'adressant à des utilisateurs issus de communautés différentes, et disposant parfois d'une expérience limitée de la modélisation.

Les outils IDEA Designer et Performer ont ainsi été présentés très tôt dans le cycle de développement du langage à des utilisateurs-testeurs, ingénieurs et experts opérationnels. Les retours effectués par ces derniers ont été pris en compte dans la suite des travaux de définition du langage : par exemple, c'est par ce biais qu'a été identifié le besoin ayant mené à la mise en place de des types et instances domaine.

L'approche par **métamodélisation** nous a été essentielle pour le bon déroulement d'une telle démarche de conception, dans la mesure où la sémantique métier est capitalisée dans le métamodèle et dans les règles associées (cohérence et validation). Il est alors possible de faire évoluer le langage et l'outil de manière aisée et réactive, sous réserve de maîtriser l'impact des modifications du métamodèle. Pour adresser la problématique de cet impact, les deux points ci-après nous apparaissent fondamentaux.

- **Concevoir les fonctionnalités outil de manière décorrélée du contenu du métamodèle** – Dans une approche où le langage de modélisation se veut évolutif, il importe de réduire autant que possible l'impact d'une modification le concernant sur les aspects IHM. Ce principe est similaire à celui de plusieurs patrons d'architecture logicielle, comme MVC (Modèle, Vue, Contrôleur) ou MVVM (Modèle, Vue, Vue-Modèle).
  Dans l'approche technologique que nous avons adoptée, ces fonctionnalités pour IDEA Designer sont prises en charge par l'environnement DSL Tools. L'environnement de modélisation est généré automatiquement à partir du métamodèle, et suite à une mise à jour de ce dernier il est donc possible très rapidement d'utiliser l'outil et d'évaluer la pertinence des modifications. Pour l'outil de simulation IDEA Performer, qui n'est pas généré par DSL Tools mais codé à la main, l'équipe de développement a eu recours au patron d'architecture MVVM pour dissocier les aspects IHM et sémantique (DSL).
- **Prévoir et gérer l'invalidation des modèles des utilisateurs** – Dans une approche où le langage est utilisé au plus tôt pour produire des modèles, ces derniers sont susceptibles de se retrouver incohérents par rapport au langage à chaque mise à jour du métamodèle. Pour ne pas perdre ces modèles, assurer leur remise en conformité devient une priorité (cf. objectif [EM1]).
  Dans certains cas, les modifications du métamodèle sont accompagnées de la mise en place de nouvelles règles visant au maintien en cohérence des modèles, qui peuvent suffire à

remettre en conformité des modèles issus de versions précédentes du langage. Cette remise en conformité peut alors se faire de manière transparente à l'utilisateur, si des règles de cohérence se chargent de modifier le modèle, ou requérir une action de sa part, si des règles de validation lui indiquent quels éléments nécessitent d'être mis à jour. Par exemple, considérons un métamodèle IDEA dans lequel il n'existe pas de lien entre les concepts *Capability* et *Process*, et sur la base duquel a été créé un modèle contenant une *Capability*. On modifie alors le métamodèle pour imposer à une *Capability* de disposer d'un *Process.* Si on définit une règle de cohérence pour créer automatiquement un *Process* pour toute *Capability* qui n'en serait pas dotée, alors cette règle rétablira la conformité du modèle existant en y ajoutant un *Process* associé à la *Capability* existante. Si on définit une règle de validation levant une erreur pour toute *Capability* qui ne serait pas dotée d'un *Process*, alors l'utilisateur serait prévenu qu'il lui faudra créer un *Process* pour mettre à jour son modèle.

Dans d'autres cas, s'il n'y a pas eu création de nouvelles règles ou si les règles créées ne sont pas suffisantes, il faut alors prévoir une démarche de remise en conformité des modèles. Cette démarche peut alors être exécutée soit manuellement, soit par le biais d'un algorithme de transformation si les modifications à réaliser sont jugées d'un volume trop important ou trop répétitives.

3.1.3. Retours d'expérience sur l'utilisabilité du langage

Tout au long des travaux présentés dans cette thèse, une attention particulière a été portée quant à la simplicité et l'expressivité du langage afin d'optimiser son utilisabilité par des experts du domaine métier pour modéliser des systèmes complexes. Cependant, il importe de noter que **le langage de modélisation ne peut assurer à lui seul cette utilisabilité**.

Les caractéristiques de l'**environnement de modélisation** permettant la création de modèles suivant ce langage sont un facteur fondamental, en particulier pour un langage visant à modéliser des systèmes complexes. En effet, une interface inadaptée ou des fonctionnalités peu ergonomiques peuvent suffire à faire réapparaître toute la complexité des systèmes modélisés.

Par conséquent, bien que les travaux de cette thèse aient porté sur le langage lui-même, il nous est apparu essentiel d'assurer que ce langage soit outillé de manière aussi utilisable que possible. En particulier, nous chercherons dans cette partie à analyser l'impact sur l'outillage de certains choix effectués pour le langage.

3.1.3.1. Quelques mots sur l'ergonomie

Le choix de l'approche DSL (plutôt que la définition d'un profil UML) s'est révélé déterminant pour la maîtrise de la représentation graphique d'un modèle et de l'interface de l'outil de modélisation. Que ce soit dans un environnement .NET (DSL Tools) ou Eclipse (Kermeta), la définition d'un DSL s'accompagne de la génération automatique d'un environnement de modélisation, et intègre la prise en compte de la définition de la représentation graphique d'un élément de modèle.

Il est ainsi possible d'adapter l'aspect graphique du modèle aux habitudes visuelles des utilisateurs, voire même de permettre à ces derniers de définir leurs propres représentations graphiques. Par exemple, les entités d'entreprises dans IDEA Designer sont représentées par des icônes, plus parlantes que des formes abstraites. Le langage IDEA devant être adaptable à différents domaines métiers ([EL2)], ces icônes sont génériques. Les experts métiers utilisant l'outil ont alors réclamé la

possibilité de définir pour un modèle un ensemble d'icônes personnalisées, et d'associer ces icônes personnalisées aux entités d'entreprise de niveau type domaine de ce modèle. Les icônes génériques ont certes été considérées plus parlantes que ne l'auraient été des simples rectangles, mais l'utilisation d'une iconographie spécifique a été perçue comme une importante plus-value pour la représentativité d'un modèle vis-à-vis du domaine métier qu'il concerne.

Si la prise en compte de différentes considérations d'ergonomie pour la représentation graphique du modèle a été ainsi aisée à mettre en œuvre, cela n'a pas toujours été le cas pour l'interface des outils générés. En effet, les possibilités de personnalisation du modeleur généré par les environnements de métamodélisation sont apparues limitées, et certaines adaptations ont requis un important effort de développement. Par exemple, DSL Tools ne permet pas nativement d'associer plusieurs diagrammes à un même modèle[24]. La présentation d'un modèle sous cette forme dans IDEA Designer, qui sera exposée dans la partie suivante, a alors nécessité pour l'équipe le développement d'une infrastructure palliant à ce manque [25][26].

3.1.3.2. Organisation du langage et interface de présentation d'un modèle

La maîtrise de la complexité du langage et la recherche de l'expression des points d'articulation majeurs a conduit à concevoir le métamodèle IDEA sur la base de plusieurs vues, regroupant les concepts et les relations décrivant un aspect particulier de l'architecture. Au cours des présentations de l'outil et des formations dispensées aux utilisateurs, nous avons constaté que cette organisation autour de quatre vues, articulées autour d'un nombre restreint de concepts-clés et reliées par des relations porteuses d'un sens métier fort (ex. *Plays*), a permis aux utilisateurs d'appréhender rapidement le périmètre couvert par le langage et d'en garder en mémoire une représentation claire et concise. Ceci a grandement facilité l'assimilation de la structure d'un modèle, et donc l'expression dans celui-ci de la connaissance métier de l'utilisateur. Par le fait qu'il induit des facilités d'appropriation du langage, on peut ainsi considérer un métamodèle bien structuré comme un **prérequis nécessaire** à la mise en œuvre des activités de modélisation.

Si elle facilite ainsi l'assimilation de la structure d'un modèle, l'organisation du métamodèle de ne peut cependant pas apporter un support suffisant à l'appropriation du contenu de ce modèle, également indispensable au bon déroulement de la modélisation. Etant donné la complexité des entreprises adressées, les modèles créés sont en effet susceptibles d'être eux-mêmes complexes et de volume important. Il importe alors que l'**organisation structurée du métamodèle** se traduise par une **présentation claire des modèles** dans l'outil de modélisation.

En particulier, il est inenvisageable de présenter le modèle de manière globale sur un diagramme unique. Il importe de définir un ensemble de points de vue sur le modèle, permettant son édition selon différentes perspectives. Il apparait alors cohérent de concevoir le mode de visualisation du modèle d'une manière qui reflète l'organisation choisie pour le métamodèle, et donc la structure du modèle. Cela revient à définir des points de vue sur le modèle, organisés selon les mêmes aspects d'architecture que les vues du métamodèle.

[24] La version disponible dans Visual Studio 2010 propose à des fins comparables le Model Bus, notamment utilisé par Visual Studio 2010 lui-même pour l'édition de diagrammes UML. Le Model Bus permet cependant de lier plusieurs DSL entre eux et d'établir des liens entre leurs diagrammes, et non de visualiser un même DSL sur plusieurs diagrammes. Pour exploiter cette fonctionnalité, il aurait alors fallu fractionner le DSL IDEA en autant de DSL que de diagrammes de modélisation souhaités.

Dans notre cas, cependant, le choix effectué en 2.3.2 d'utiliser dans le métamodèle un même concept pour représenter les types et les instances domaine amène un nécessaire ajustement entre le périmètre des vues du métamodèle et celui des points de vue sur le modèle. En effet, il apparait indispensable de présenter dans l'outil de modélisation les éléments de niveau type et instance domaine de manière séparée, afin de permettre à l'utilisateur de faire facilement la distinction entre les niveaux de modélisation auxquels ils correspondent. Dans IDEA Designer, les éléments de niveau instances sont donc regroupés dans un point de vue qui leur est dédié.

Les points de vue sur le modèle tels que présentées à l'utilisateur dans IDEA Designer sont alors au nombre de cinq, définis de la manière suivante :

Point de vue	Vue de métamodèle	Périmètre
COMMONS	COMMONS	Produits & modèle de données exploités par l'entreprise
ENTERPRISE CAPABILITY MAP	CAPABILITY MAP	Capacités, services, processus, cellules capacitaires
ENTERPRISE COLLABORATIONS	ROLES & COLLABORATIONS	Schémas de collaborations entre rôles de niveau type domaine
ENTERPRISE RESOURCES & ORGANIZATION	ENTERPRISE ORGANIZATION	Organisation des entités d'entreprise de niveau type domaine
ENTERPRISE CONFIGURATION	∅	Collaborations et entités de niveau instance domaine

Tableau 6 - Points de vue sur le modèle

Le périmètre défini par chacun de ces points de vue reste également trop important pour être représenté sur un diagramme unique. Par exemple, le modèle qui sera présenté en 3.2 compte environ 200 capacités (définissant chacune un processus plus ou moins complexe), regroupés en une quarantaine de cellule capacitaires. Un diagramme unique présentant le point de vue ENTERPRISE CAPABILITY MAP d'un tel modèle compterait donc, même sans compter les activités composant les processus associés à chacune des capacités, environ deux cents éléments. Ces éléments étant de plus reliés les uns aux autres, la lecture de ce diagramme s'avèrerait difficile et fort peu ergonomique. Il apparait ainsi essentiel de raffiner les points de vue sur le modèle, en définissant pour chacun d'eux un ensemble de types de diagrammes présentant un sous-ensemble des éléments. A titre d'exemple, l'ensemble des diagrammes de chacun des points de vue définis par l'équipe IDEA pour IDEA Designer est disponible en Annexe 6.

Il importe ici de noter que bien que le résultat soit visuellement comparable, cette approche diffère de celle adoptée par les Architecture Frameworks ou d'autres langages de modélisation comme UML. Pour ces derniers, les diagrammes définissent des éléments de modèle, et l'utilisateur peut exprimer les liens entre les éléments appartenant à des diagrammes différents. Dans notre cas, les différents diagrammes d'IDEA Designer ne constituent qu'une représentation graphique d'une information contenue dans un modèle, et non la définition de cette information. En particulier, cela signifie que la destruction d'un diagramme IDEA n'entraîne pas la destruction des éléments de

modèle qu'il présente, alors que les éléments d'un modèle issu d'un Architecture Framework sont agrégés dans le produit qui les définit.

En support à la démarche de conception, l'avantage majeur de cette approche de la présentation du modèle dans l'outil est de permettre, en fonction des besoins remontés par les utilisateurs, de rajouter facilement des diagrammes à l'outil sans impact sur le langage. En particulier, la présentation du modèle telle qu'introduite dans cette partie, basée sur les vues du métamodèle, ne contient pas de diagramme qui présenterait le modèle selon un point de vue orthogonal aux vues du métamodèle. Par exemple, il serait intéressant pour l'étude de l'architecture de disposer d'un diagramme présentant une vue « à plat » des relations de services existant entre les entités de niveau instance domaine du fait de leur participation aux collaborations de niveau instance, ou encore d'un diagramme de séquence présentant l'ordonnancement des processus mis en œuvre par les différents rôles d'une collaboration. Dans les deux cas, l'information nécessaire à la construction des diagrammes est intégralement présente dans le modèle. A la date de rédaction de cette thèse, l'ajout d'un certain nombre de tels diagrammes globaux dans IDEA Designer est à l'étude suite à des besoins remontés par les utilisateurs.

3.1.3.3. Règles du langage et fonctionnalités d'assistance à la modélisation

Au-delà des aspects d'interface utilisateur, l'outil est également le dépositaire des fonctionnalités de modélisation, et il parait ainsi naturel qu'il contribue à l'objectif de modélisation assistée ([EL4]). Cette contribution peut intervenir selon les deux axes identifiés en 3.1.2 pour la classification des règles du langage, en complément de ces dernières.

Comme annoncé en 1.3.1.3, nous avons cherché à ce que la mise en place dans le langage des fonctionnalités d'expression de l'adaptabilité de l'entreprise complexifie le moins possible la définition d'un modèle. Les fonctionnalités outil d'assistance à la modélisation constituent un moyen privilégié pour atteindre ce but, dans la mesure où elles permettent de masquer toute complexité inutile à l'utilisateur.

Tout comme les règles passives, l'environnement de modélisation peut superviser les actions de l'utilisateur et lui interdire celles qui seraient en contradiction avec des contraintes d'architecture. L'intérêt de telles fonctionnalités outil par rapport aux règles de cohérence réside dans le fait que ces dernières sont déclenchées *après* les actions de l'utilisateur, alors que l'outil peut intervenir *avant* et améliorer l'expérience utilisateur.

Par exemple, considérons l'établissement de relations *Delegates* entre deux *Endpoints*. Dans IDEA Designer, ces relations peuvent se construire en utilisant un outil de connexion entre la source et la cible attendues dans un diagramme dédié à l'édition de l'implémentation d'une collaboration de niveau type domaine. En termes de contraintes, la source de la relation de délégation doit être un *Endpoint* appartenant à l'interface d'une collaboration, et la cible doit être un *Endpoint* de même type (ex. envoi d'un produit), de caractérisation similaire (ex. même produit envoyé) et appartenant à l'un des rôles ou à l'une des interactions impliqués dans la collaboration. Le tracé d'une telle relation dans l'outil peut alors se dérouler des deux manières suivantes :

- **Cas 1** – Considérons que la contrainte sur la source et la cible d'une relation *Delegates* n'est implémentée que dans une règle de cohérence. A partir d'un *Endpoint* source donné, l'outil de connexion généré par l'environnement de métamodélisation autorise la sélection de tout élément cible conforme à la relation du métamodèle, à savoir l'ensemble des *Endpoints* de

même type. Il est ainsi possible à l'utilisateur de sélectionner pour cible un *Endpoint* appartenant à la même interface que la source, et/ou qui soit de caractérisation différente. Dans ce cas, la règle de cohérence déclenchée suite à la sélection refusera l'établissement de la relation en remontant un message d'erreur comme illustré sur la Figure 38.

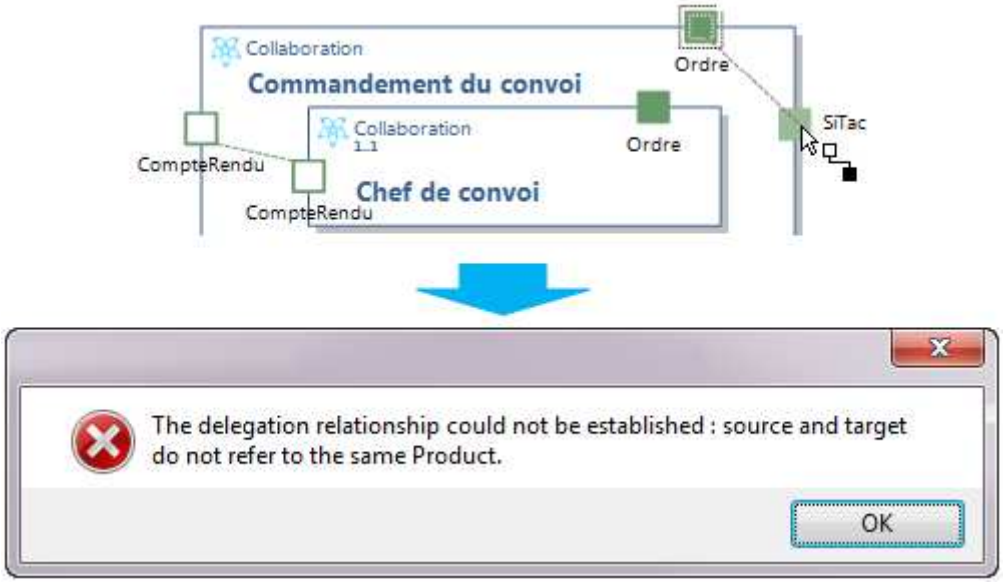


Figure 38 - Illustration des fonctionnalités d'assistance à la modélisation (1)

- **Cas 2** – Si l'on a veillé à ce que l'outil prenne directement en compte la contrainte d'architecture plutôt que de ne tenir compte que du métamodèle, alors l'outil de connexion ne permet de cibler que les *Endpoints* de même type, de caractérisation similaire et appartenant à un rôle impliqué dans la collaboration portant la source comme illustré sur la Figure 39. Il est ainsi non seulement impossible d'établir une relation en contradiction avec la contrainte, mais en plus l'éditeur offre une aide à la décision en excluant d'office les choix incohérents.

Figure 39 - Illustration des fonctionnalités d'assistance à la modélisation (2)

L'outil peut également mettre à disposition de l'utilisateur des « raccourcis » de modélisation. Le principe de ces fonctionnalités est alors de prendre en charge plusieurs actions sur le modèle suite à une seule action utilisateur. Comme les règles actives, ces fonctionnalités permettent d'assurer la complétude ou la conformité de la structure du langage, ainsi que de réduire le nombre d'actions à effectuer par l'utilisateur. Dans le cadre d'une approche de prototypage rapide basé sur des boucles modélisation / simulation d'une durée de quelques heures, ce dernier point représente une considération d'importance non négligeable.

Dans le cadre de l'exemple du connecteur des relations *Delegates*, une telle fonctionnalité a été implémentée pour permettre à l'utilisateur de tracer le connecteur à partir d'un *Endpoint* source de

l'interface d'une collaboration vers le rôle sensé porter l'*Endpoint* cible. C'est alors l'outil qui prend en charge la sélection de la cible. Si l'interface du rôle ciblé contient un *Endpoint* compatible, alors la relation est établie vers lui. Sinon, l'outil prend en charge la création sur l'interface du rôle ciblé d'un nouveau *Endpoint* du même type que la source, le caractérise de manière identique, puis établit la relation vers ce nouvel *Endpoint* comme illustré sur la Figure 40.

Cette fonctionnalité permettant dans le deuxième cas d'effectuer en une seule trois actions (création d'un *Endpoint* cible, caractérisation du *Endpoint*, création de la relation de délégation), nous avons constaté qu'elle a été très bien accueillie et massivement utilisée par les utilisateurs.

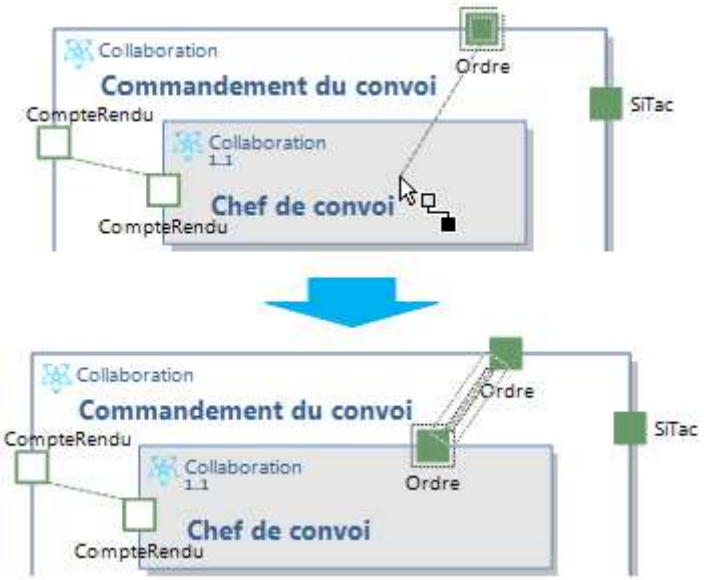


Figure 40 - Illustration des fonctionnalités d'assistance à la modélisation (3)

L'intégration de telles fonctionnalités d'assistance à la modélisation requiert cependant une attention particulière au niveau de la gestion du projet logiciel de l'outil de modélisation, afin d'y éviter toute redondance inutile et compliquant la maintenance. Il importe en effet de capitaliser les tests utilisés pour vérifier la conformité du modèle vis-à-vis des contraintes, par exemple en les encapsulant dans des méthodes auxiliaires. Une telle capitalisation n'apporterait cependant pas de perte d'information dans le langage, les méthodes auxiliaires étant *a priori* portées par les classes décrivant les concepts du métamodèle. Par exemple, tester si les deux *Endpoints* concernés par une relation de délégation sont de même type et de caractérisation similaire peut faire l'objet d'une méthode `IsCompatibleWith()`, portée par la classe *Endpoint*.

3.2. Une application métier : le modèle de commandement

3.2.1. Contexte

3.2.1.1. Caractéristiques du modèle de commandement

Le modèle qui fera l'objet de l'ensemble de cette partie, dit « modèle de commandement », a été réalisé dans le cadre d'une affaire industrielle adressant le contexte métier de l'Armée de Terre.

L'entreprise considérée est l'organisation en charge du commandement des forces déployées sur un théâtre d'opérations. Cette organisation est composée d'un certain nombre d'états-majors[25], eux-mêmes organisés en cellules correspondant à des domaines métier différents (ex. logistique, génie…). Dans la suite de ce chapitre, nous utiliserons l'acronyme « EM » pour faire référence à un état-major dans l'organisation considérée.

Cette entreprise vérifie les critères de Maier et le critère DGA pour les systèmes de systèmes :

- **Indépendance opérationnelle** – Les différents EM peuvent et doivent pouvoir fonctionner indépendamment les uns des autres,
- **Indépendance managériale** – Les différents EM appartiennent à des chaînes managériales variées, que ce soit au sein d'une même nation (ex. coordination entre l'Armée de Terre et l'Armée de l'Air) ou entre des nations différentes dans le cadre d'une coalition,
- **Développement évolutionniste** – Les EM doivent pouvoir interagir avec de nouveaux partenaires au gré des évolutions des missions et de la situation (ex. intégration d'une nouvelle nation dans une coalition), et en interne peuvent intégrer de nouvelles unités et/ou de nouveaux systèmes au gré des besoins,
- **Comportement émergent** – Ce n'est qu'en combinant les capacités des différents EM que l'ensemble de la force peut réaliser ses missions,
- **Distribution géographique** – Les EM sont dispersées sur le théâtre d'opération et le ou les territoires nationaux,
- (Critère DGA) **Objectif commun** – Les EM poursuivent une mission commune.

Le modèle a été réalisé en phase d'analyse fonctionnelle de l'organisation existante. Son objectif est la capture et la formalisation des processus et des échanges d'information mis en œuvre, afin de constituer une base pour des travaux d'architecture ultérieurs.

L'intérêt de ce modèle pour l'évaluation des travaux de thèse porte sur plusieurs points :

- **Son périmètre** – Il utilise tous les travaux de thèse disponibles dans la version livrée. Il inclut notamment l'étude de l'adaptabilité de l'entreprise concernée, sans pour autant que cette étude ne constitue l'objectif principal.
- **Sa démarche de mise en œuvre** – Le modèle a été élaboré sur la base de boucles rapides modélisation / simulation conformément au contexte initial guidant les travaux.
- **L'importance de l'implication des experts opérationnels dans sa réalisation** – L'objectif du modèle étant la capture de l'existant, la conformité des processus modélisés par rapport au

[25] « Un état-major est un ensemble de personnel (officiers, sous-officiers et militaires du rang formés aux tâches d'état-major) et d'équipements, mis à la disposition d'un chef pour lui permettre de commander une force, une grande unité ou une formation de niveau inférieur ». Source : Centre de Doctrine d'Emploi des Forces (CDEF)

métier est essentielle. Il était donc nécessaire d'impliquer les experts opérationnels autant que possible, et d'assurer que le formalisme de modélisation ne représente pas un frein à l'expression de leur connaissance dans le modèle.

3.2.1.2. Quelques données

Le Tableau 7 ci-dessous présente les principales données de volume du modèle de commandement, permettant d'estimer la complexité du modèle (en particulier à l'exécution). Il importe de noter que ces valeurs sont des **valeurs moyennes** constatées pour différentes instanciations du modèle. Les différences entre les déploiements résultants portent essentiellement sur les collaborations et les organisations d'entreprise, dans le but d'étudier des problématiques opérationnelles distinctes.

Nombre d'entités instanciées	50
Nombre de collaborations instanciées	150
Nombre de processus modélisés	250
Nombre moyen de processus exécutés de manière concurrente à l'exécution	50

Tableau 7 - Données de volume du modèle de commandement

En termes d'effort, la réalisation de ce modèle a représenté au total **4 mois de travail pour 1 architecte système**, en collaboration ponctuelle avec des experts opérationnels apportant leur connaissance du métier. Sur la base de l'expérience de l'industriel et étant donné la complexité du domaine et la taille du système de systèmes à modéliser, cet effort est très bon par rapport aux bénéfices qu'il apportera dans la suite des travaux.

Les 4 mois incluent :

- La **compréhension du domaine** (estimée à 1/3 du temps),
- La **réflexion sur les principes de modélisation** les plus adaptés à la problématique (estimée à 1/6 du temps),
- La **modélisation** elle-même (estimée à 1/3 du temps),
- La **validation du modèle** par présentation dynamique (simulation) aux opérationnels (estimée à 1/6 du temps).

La répartition des efforts fait apparaître que la compréhension du domaine est une activité chronophage. Une telle situation peut être due à de multiples facteurs : domaine complexe et peu formalisé, effort d'acculturation nécessaire si l'architecte n'a initialement pas connaissance du milieu, etc. Par rapport à la démarche d'architecture, cette constatation renforce l'intérêt d'associer les experts du domaine à la réalisation du modèle, voire de leur confier cette tâche pour les analyses fonctionnelles.

3.2.2. Analyse de la pertinence des contributions vis-à-vis du modèle

3.2.2.1. Adéquation des contributions à l'entreprise modélisée

Dans la partie 3.2.1.1, nous avons montré que l'entreprise considérée pour le modèle de commandement vérifie les critères retenus pour qualifier les systèmes de systèmes. Les systèmes de systèmes représentent en effet une part importante des entreprises adressées par la démarche dans

laquelle nous nous positionnons, et le langage doit permettre de prendre en compte les différents critères à la définition d'un modèle.

Les moyens permettant d'adresser l'**indépendance opérationnelle**, l'**indépendance managériale** et la **distribution géographique** sont offerts par des travaux développés en dehors cadre de cette thèse.

L'indépendance opérationnelle et managériale est supportée par les différentes relations dans les organisations des *Enterprise Entities*. En particulier, les relations transverses et le concept de *COI* permettent de relier au sein d'une même entreprise des entités appartenant à des chaînes hiérarchiques différentes.

Les aspects géographiques ne sont pas abordés dans le DSL IDEA, à l'exception d'une propriété de localisation sur les entités. Cette localisation n'est exploitée que pour la simulation. On peut ainsi considérer que la distribution géographique est supportée par la possibilité, dans l'outil de simulation, de positionner les *Enterprise Entity* à différents endroits d'une vue cartographique.

En termes de caractéristiques des systèmes de systèmes, les contributions de cette thèse concernent les **comportements émergents** et l'**objectif commun,** ainsi que le **développement évolutionniste**. Les deux premiers apparaissent avec la **structure récursive des organisations de rôles** et le concept de collaboration. En effet, ce sont les interactions entre plusieurs rôles qui permettent de réaliser les attentes du rôle de niveau supérieur auquel ils collaborent. Les rôles d'une même collaboration poursuivent un même but de réalisation de cette collaboration, en termes de fourniture et de consommation de services ou de produits.

Le développement évolutionniste se traduit dans la flexibilité apportée par les **configurations de rôles**. Comme présenté en 2.4.1.2, le principe et la mise en œuvre des configurations de périmètre libre permet de faire intervenir dans une collaboration tout rôle respectant une interface donnée. Il est ainsi possible pour un nouveau rôle de venir contribuer à une collaboration existante.

3.2.2.2. Adéquation des contributions à la complexité du modèle

La réalisation du modèle de commandement a tout d'abord permis de valider l'intérêt de disposer d'un modèle d'architecture simulable. La simulation représente en effet un mode de visualisation du modèle bien plus expressif et interactif que la vue statique en diagrammes, et se révèle ainsi un excellent support de discussion. En particulier, elle a permis à des experts opérationnels non familiers avec le formalisme d'appréhender le périmètre et le contenu du modèle très rapidement, c'est à dire en quelques minutes. Ainsi, des séances de quelques heures se sont révélées suffisantes pour obtenir des retours quant à la conformité des processus modélisés par rapport au métier, et/ou consolider le modèle.

La simulation a été également régulièrement utilisée à des fins de débogage du modèle, dans la mesure où elle permet rapidement de s'assurer de la synchronisation et du bon enchaînement des processus modélisés et de corriger ainsi les erreurs de saisie. Nous avons alors constaté qu'il importe d'utiliser la simulation dans cette optique au plus tôt lors de la conception du modèle, et de préférence sur des sous-ensembles du modèle afin de minimiser la complexité du comportement à vérifier.

Le modèle a également permis de confirmer l'adéquation du langage (et des outils) à la complexité et au volume des entreprises ciblées, en particulier des systèmes de systèmes. Si d'un point de vue

technique le volume du modèle de commandement (voir partie précédente) ne s'est pas révélé un frein aux outils, du point de vue d'un utilisateur ou d'un observateur il atteint une complexité importante vis-à-vis de ce qu'il est possible d'appréhender rapidement sur un diagramme de modélisation ou lors d'une simulation. A titre d'exemple, la Figure 41 ci-après présente la vue globale des *Enterprise Entity* de niveau instance domaine dans IDEA Designer.

Figure 41 – Vue globale des *EnterpriseEntity* de niveau instance domaine du modèle de commandement

3.2.3. Mise en œuvre des contributions

3.2.3.1. Types / instances domaine et configurations dans le modèle de commandement

Les concepts et relations composant le langage se sont révélés adaptés à la modélisation de l'entreprise considérée. En particulier, le principe types / instances domaine a été fondamental pour pouvoir capitaliser des aspects communs à plusieurs éléments d'organisation et éviter ainsi de complexifier inutilement le modèle.

Par exemple, les relations hiérarchiques entre les EM concernés par le modèle de commandement peuvent être représentées de manière simplifiée à la manière d'un arbre ternaire : l'EM de niveau hiérarchique noté « EM N3 »[26], compte trois EM subordonnés (« EM N4-1 », « EM N4-2 » et « EM N4-3 »), et ainsi de suite jusqu'au niveau hiérarchique 6. Tous les EM à un niveau hiérarchique donné (ex. tous les « EM N6-… ») sont similaires en termes de capacités, de missions et de décomposition

[26] Dans le domaine militaire, l'EM de niveau hiérarchique conventionnellement noté « 3 » correspond au commandement d'une brigade, c'est-à-dire d'un volume équivalent à celui des forces françaises de l'Armée de Terre engagées en Afghanistan en 2010.

interne. Le principe types / instances domaine a permis de capitaliser les éléments communs aux EM d'un niveau hiérarchique donné dans une même *Enterprise Entity* de niveau type domaine.

La Figure 42 présente une capture d'écran de l'organisation des *Enterprise Entity* de niveau type domaine, avec un seul ensemble d'*Enterprise Entity* pour décrire la composition de chacun des EM aux niveaux hiérarchiques de l'organisation considérée (de 3 à 6). Les relations entre les différents niveaux sont dotées d'une cardinalité [3..3] : l'instanciation de l'entité de plus haut niveau, « N2 » (représentant la métropole) donnera lieu à la constitution automatique au niveau instances domaines d'une arborescence ternaire complète pour les niveaux 3 à 6.

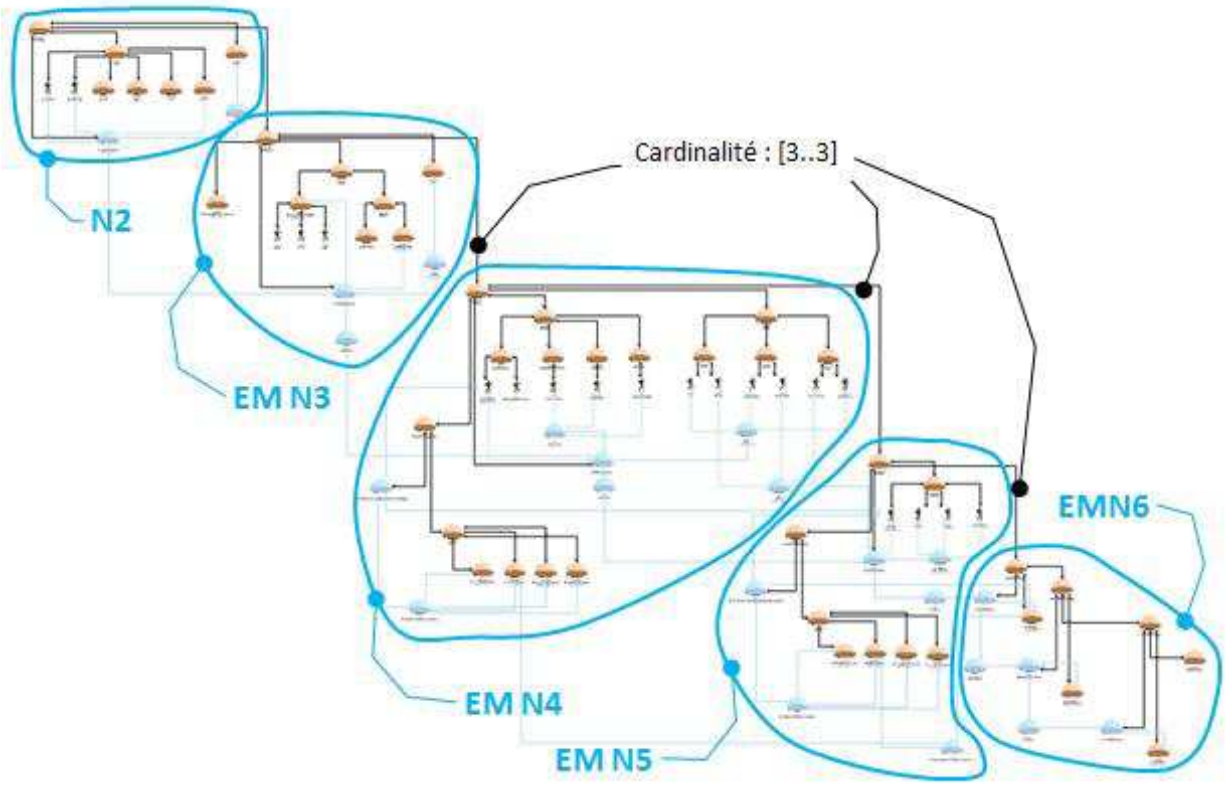


Figure 42 - Organisation des entités de niveau type domaine dans le modèle de commandement

En termes de complexité et de lisibilité, la Figure 42 est à comparer avec la Figure 41 présentant les *Enterprise Entities* correspondantes au niveau instance domaine. Du point de vue de l'utilisateur, le travail au niveau des types domaine est ainsi bien plus simple qu'au niveau instances.

La méthode de travail adoptée pour l'élaboration du modèle de commandement a été d'instancier les organisations très tôt, puis de concentrer l'effort de modélisation sur le niveau type domaine. Les instances domaine ont été alors intégralement constituées par les règles de maintien en cohérence types / instances présentées en 2.3.4 au fil des mises à jour de leurs types.

Dans une optique similaire, le concept de *Collaboration* a également été considéré comme d'un intérêt majeur, par le fait qu'il permet de décrire les aspects fonctionnels et comportementaux (schémas d'interactions entre processus) de manière décorrélée de l'organisation spécifique qui les met en œuvre.

Dans le modèle de commandement, les processus et les interactions (services et échanges d'information) mis en œuvre par les EM des niveaux hiérarchiques N3, N4 et N5 sont similaires. Si les

aspects comportementaux représentés par les processus et les interactions étaient directement alloués aux *Enterprise Entity*, cela requerrait de les définir trois fois (une fois par niveau). Dans le modèle, les processus et les schémas d'interactions sont capitalisés au niveau d'une Collaboration unique de niveau type domaine (notée « Collaboration N »), instanciée une fois pour chacune des *Enterprise Entity* de niveau instance domaine correspondant aux niveaux N3, N4 et N5. Ces instances sont maintenues automatiquement en cohérence avec leur type domaine.

Ce principe est schématisé par la Figure 43 pour les niveaux N3 et N4.

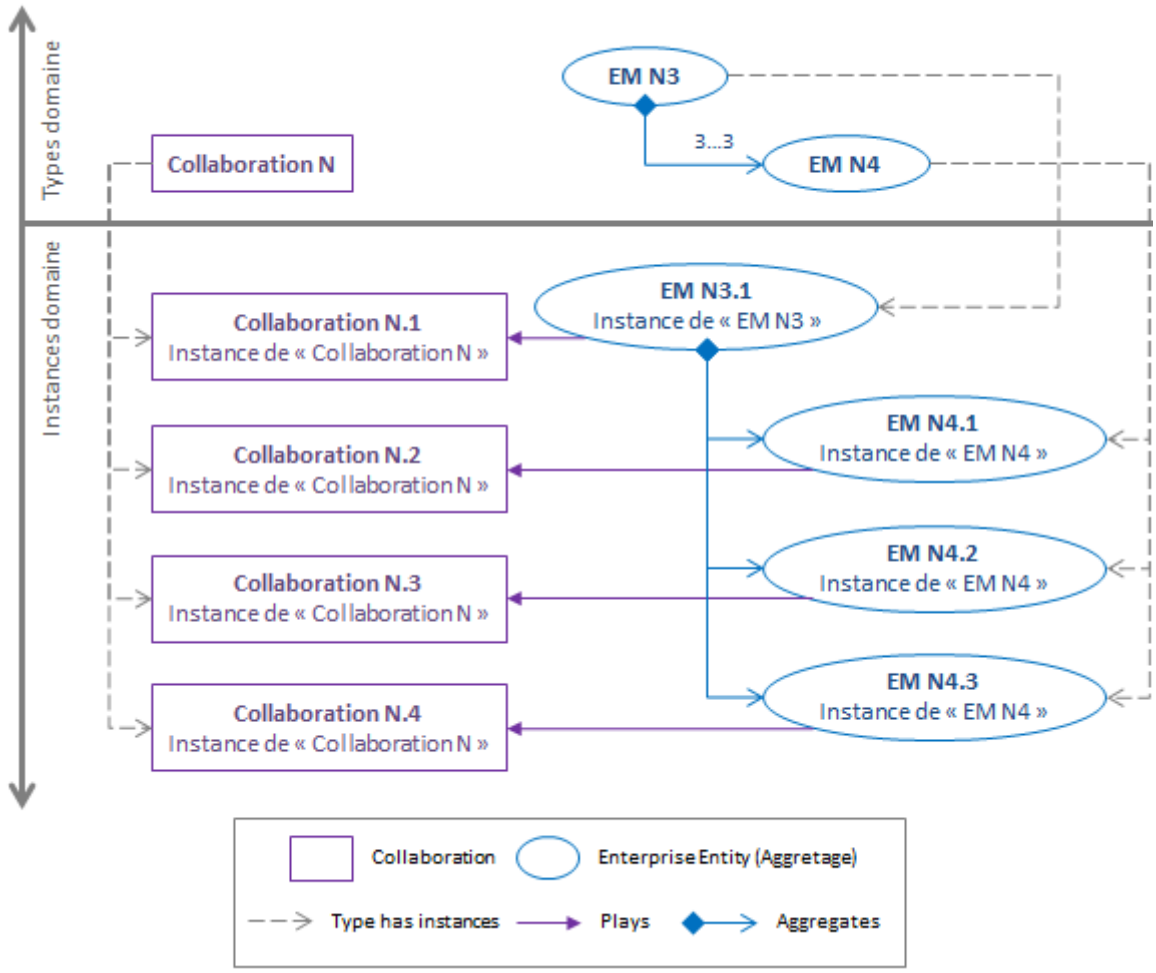


Figure 43 – Principe de modélisation des collaborations au niveau type et instance domaine dans le modèle de commandement

La capitalisation des schémas d'interactions au niveau d'une collaboration unique (« Collaboration N ») entraîne une problématique non triviale quant à la manière de modéliser ces schémas d'interactions. En effet, chacune des instances domaine de cette collaboration doit pouvoir échanger avec ses niveaux hiérarchiques subordonnés et supérieurs, eux-mêmes représentés par des instances domaine de la « Collaboration N ».

C'est le principe d'interchangeabilité des rôles qui a été mis en œuvre pour répondre à cette problématique : dans la « Collaboration N » a été défini un rôle « Niveau supérieur », doté de l'interface nécessaire pour communiquer avec un subordonné (de manière simplifiée, envoyer des ordres et recevoir des comptes-rendus). L'interface de la « Collaboration N » étant compatible de

celle du « Niveau supérieur », il est alors possible après instanciation de lier les collaborations des différents niveaux en venant à chaque niveau remplacer le rôle « Niveau supérieur » par la collaboration « Collaboration N » associée à l'état-major de niveau supérieur.

Ce principe est illustré sur la Figure 44 pour deux EM instances de niveau N2 et N3 :

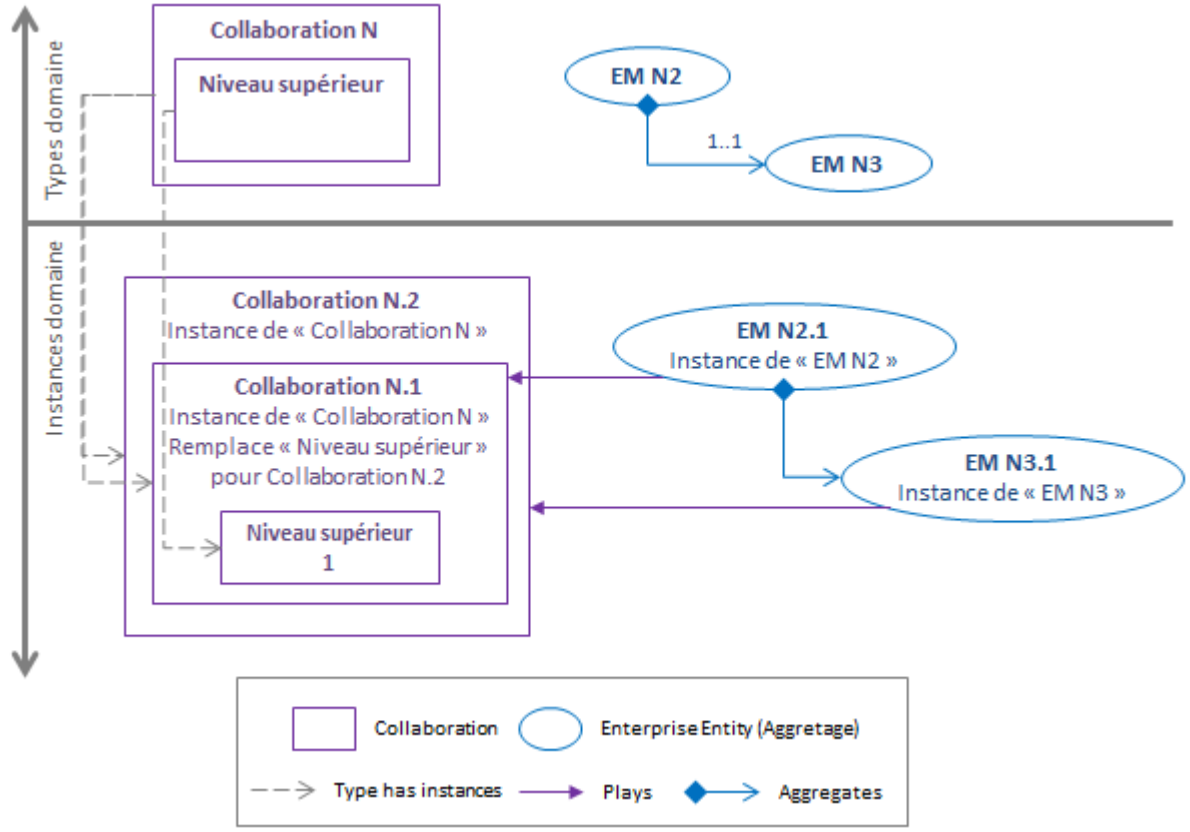


Figure 44 – Principe de mise en œuvre de l'interchangeabilité des rôles dans le modèle de commandement

3.2.3.2. Modification dynamique du modèle

Dans le cadre de la logistique de théâtre, les processus métiers concernés par le périmètre d'étude du modèle de commandement comprennent l'étude de la constitution d'une communauté d'intérêt (COI), visant à superviser l'évacuation d'un matériel endommagé dans une zone de combat. La COI n'est formée qu'en cas de nécessité mais la procédure d'évacuation qu'elle doit mettre en œuvre est connue et établie. Il n'y a pas d'acteurs dédiés à cette COI, ceux qui viendront y prendre en charge la procédure d'évacuation seront choisis parmi les acteurs qui déroulent la mission nominale, en fonction de leur charge et de leur disponibilité.

Cette situation opérationnelle se traduit dans le modèle de commandement par la création **en cours de simulation** d'une *EnterpriseEntity* « COI d'évacuation exceptionnelle », de niveau instance domaine, dont les membres vont ensuite dérouler des processus simulant la supervision de l'évacuation. Plus précisément, la simulation déroule les étapes suivantes :

- Sur arrivée d'un évènement particulier (simulant la demande d'évacuation), instanciation de la « COI d'évacuation exceptionnelle » et d'une collaboration associée «Supervision d'une évacuation exceptionnelle », décrivant les procédures et les rôles à mettre en œuvre,
- Sélection des acteurs les plus à même d'assurer ces rôles, parmi les *EnterpriseEntity* de niveau instance,

- Association des acteurs aux rôles, ajout des acteurs en tant que membres de la COI,
- Déclenchement des processus des acteurs.

Ces différentes étapes, exploitant au niveau du modèle le principe de types et instances domaines, sont orchestrées par le biais d'une fonctionnalité d'extensibilité des modèles intégrée au niveau des outils IDEA Studio. Cette fonctionnalité permet d'associer aux activités de processus un comportement personnalisé représenté par une méthode C#. Ces fonctions d'extension offrent un vaste ensemble de possibilités, comme par exemple la journalisation de la simulation ou l'implémentation de règles métier relevant du domaine spécifique du modèle. Comme elles ont on contrôle total sur le modèle, ces fonctions permettent de décrire l'ensemble des comportements requis pour simuler l'évacuation exceptionnelle.

Chapitre 4. Conclusions et perspectives

La complexité croissante des Systèmes de Systèmes et autres grands systèmes civils et militaires constitués par une fédération d'acteurs (« entreprises ») pose de nouvelles problématiques de conception et de réalisation. Cette complexité, induite par des structures de management toujours plus sophistiquées et un cycle de vie long, doit être maîtrisée au plus tôt dans la conception des entreprises. Cette maîtrise permettra à l'ensemble des intervenants de la conception d'identifier les points clés de l'entreprise et de prendre confiance en sa capacité à atteindre ses objectifs. En particulier, il importe de savoir estimer les capacités de l'entreprise à s'adapter à des situations imprévues ou exceptionnelles, par exemple par le biais d'une reconfiguration, afin d'assurer ses missions en toutes circonstances.

Nous nous intéressons aux apports d'une démarche dirigée par les modèles pour la maîtrise de la complexité de l'entreprise en phase de conception et l'évaluation de son adaptabilité. En particulier, il apparaît important d'impliquer des experts du domaine dans la réalisation des modèles d'architecture. Cette implication permet de s'assurer de la bonne compréhension du besoin autant que de la cohérence de l'architecture réalisée par rapport au domaine métier. Nous recherchons alors à mettre en œuvre une approche de conception d'architecture impliquant des experts opérationnels, basée sur un modèle d'architecture simulable permettant des boucles courtes entre activités de modélisation et de simulation (de l'ordre d'une journée).

En support à une telle démarche, nous avons cherché à enrichir un langage de description et de simulation d'architecture d'entreprises afin de le doter de moyens d'expression de l'adaptabilité de l'architecture. Afin de mettre en exergue les points de l'architecture contribuant à l'adaptabilité de l'entreprise (points d'articulation), le métamodèle au cœur de ce langage est organisé autour de trois vues principales : les capacités et les processus déroulés (le *quoi*), l'organisation des entités composant l'entreprise (le *qui*) et les rôles et les collaborations mis en œuvre pour permettre aux entités de prendre en charge les processus (le *comment*).

Afin d'exploiter ces points d'articulation, nous avons distingué deux catégories d'entités d'entreprises et de rôles : les *instances domaine* peuplant la simulation et les *types domaine* décrivant les caractéristiques communes à un ensemble d'instances domaine de même nature (ex. les navires d'une même classe). Nous avons de plus intégré au langage des mécanismes permettant de modéliser des configurations d'entités et de rôles, afin d'évaluer leur reconfiguration.

Les travaux réalisés ont ainsi permis d'enrichir le langage selon les axes suivants :

- Gestion des types et instances domaine,
- Configurations et possibilités de reconfiguration des organisations d'entités et des collaborations de rôles,
- Maintien en cohérence et validation des modèles.

Le langage, incluant les contributions de cette thèse, a été mis en œuvre au cœur d'une suite d'outils de modélisation et de simulation. L'ensemble a été expérimenté dans le cadre d'une affaire industrielle, qui a permis de confronter les réalisations à un modèle métier complexe. A la date de soutenance de cette thèse, la suite d'outils est toujours utilisée par l'industriel.

La modification d'un modèle au cours de la simulation, permise par l'introduction des mécanismes de configuration dans le langage, est une fonctionnalité d'un grand intérêt pour les utilisateurs. Elle est au cœur des perspectives d'évolution du langage et des outils. Nous allons ainsi faciliter la définition des comportements de reconfiguration en cours de simulation. Basée sur des concepts

simples (activités dans les processus métier), cette définition pourra être mise en œuvre par l'ensemble des utilisateurs, y compris opérationnels.

Au-delà du langage et des outils présentés, les expérimentations et les résultats sur le sujet de l'adaptabilité et de la reconfiguration des Systèmes de Systèmes offrent à l'industriel des mécanismes pour l'élaboration lignes de produits (PMS – *Package Modular Solutions*). Ces résultats contribuent également à la gestion des systèmes tactiques, en particulier en ce qui concerne la supervision de leur reconfiguration en cours de mission.

Enfin, nous souhaitons nous inspirer de ces résultats afin d'étendre les *Architecture Frameworks*, pour leur permettre de décrire l'adaptabilité et la reconfiguration des architectures et les simuler.

Annexes

Annexe 1 – Informations complémentaires concernant les propriétés domaine et autres caractéristiques communes aux concepts du langage IDEA

Dans le DSL IDEA, les caractéristiques communes (métadonnées et propriétés domaine ont été regroupées au niveau d'un concept *Prototype*, dont héritent tous les concepts représentant les différents constituants d'architecture. Le nom et les métadonnées sont définis comme des attributs de ce concept. L'identifiant n'est pas défini dans le métamodèle, mais est géré automatiquement par l'environnement DSL Tools de manière transparente pour l'utilisateur[27].

Les propriétés domaine sont quant à elles représentées par un concept *Property*. Le type d'une propriété domaine peut être soit un type primitif (String, Integer…) ou une énumération, soit une structure composite à base de types primitifs. Par exemple, les coordonnées géographiques peuvent se représenter sous la forme d'une structure composite formée de trois `Double`, correspondant à la latitude, la longitude et l'altitude. Comme illustré sur la Figure 45, le type d'une propriété domaine est représenté dans le DSL IDEA par un concept abstrait *ValueType*, dont héritent un concept *PrimitiveType* (correspondant aux types primitifs .NET), un concept *EnumType* (correspondant aux énumérations) et un concept *ComplexType* (correspondant aux types composites).

La valeur d'une propriété est représentée sur le concept *Property* par un attribut *SerializedValue* de type `String`. Pour des raisons de simplicité, cette valeur n'est pas stockée en tant qu'élément de modèle, mais est sérialisée / désérialisée à la demande directement dans le code xml du modèle. La valeur est désérialisée pour être présentée à l'utilisateur (pour son édition), et sérialisée à sa modification.

On notera que l'expressivité des propriétés domaine ainsi représentées dans le métamodèle reste limitée, en particulier par le fait que les références vers les objets d'architecture ne sont pas supportées. Par exemple, il n'est pas possible d'exprimer le fait qu'un radar dispose d'une propriété domaine « Cible » qui référence une *EnterpriseEntity* détectée.

[27] Il s'agit d'un UUID (Universal Unique IDentifier), similaire à celui du format d'échange xmi.

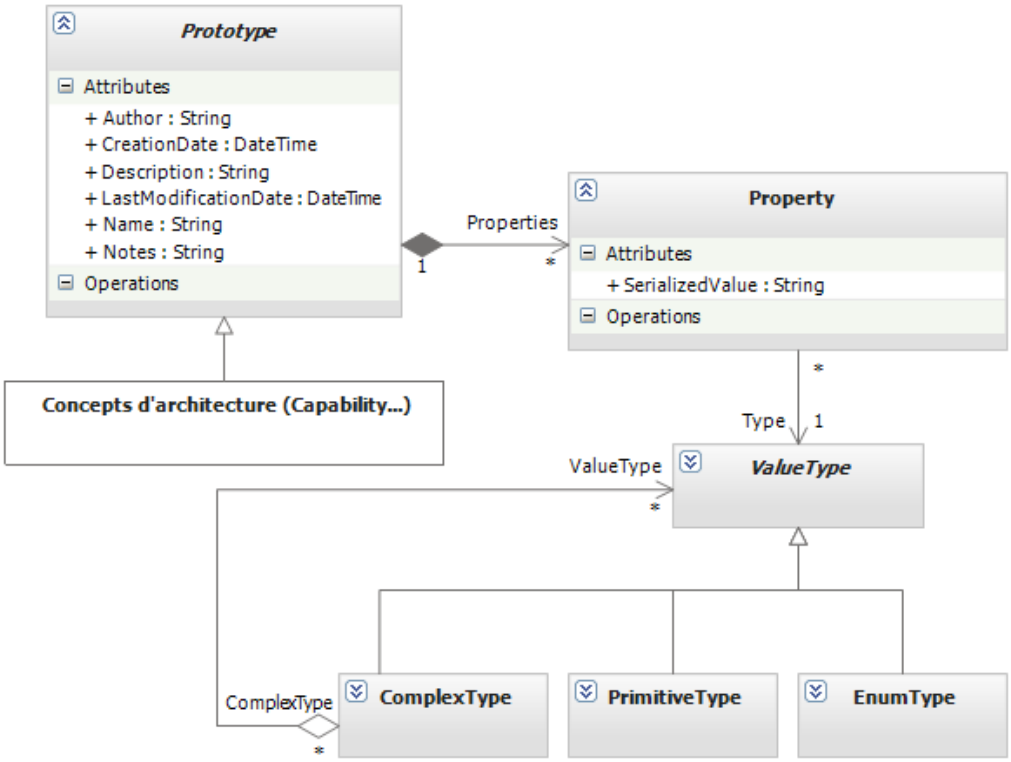


Figure 45 – Concepts *Prototype* et *Property* dans le DSL IDEA

Annexe 2 – Erreurs de validation pour la cohérence d'un déploiement

Concept	Propriété	Erreur	Description
Aggregate	*Aggregated Entities*	Multiplicité incohérente	Levée quand le nombre d'Entités Instances agrégées est incompatible avec la multiplicité définie au niveau des Types.
		Entité agrégée incohérente	Levée quand un Agrégat instance agrège une Entité dont le type n'est pas agrégé dans le type de cet Agrégat.
COI	*Gathered Entities*	Multiplicité incohérente	Levée quand le nombre d'Entités Instances participantes est incompatible avec la multiplicité définie au niveau des Types.
		Participant incohérent	Levée quand une COI instance contient une Entité dont le type n'est pas contenu dans le type de cet COI.
Physical Enterprise Entity	*Contained Entities*	Multiplicité incohérente	Levée quand le nombre d'Entités Instances embarquées est incompatible avec la multiplicité définie au niveau des Types.
		Entité embarquée incohérente	Levée quand une Entité Physique instance contient une Entité dont le type n'est pas contenu dans le type de cette Entité Physique.
	Hosted Entities	La définition des relations Hosts au niveau type ne doit pas être une restriction pour le niveau instance : dans certains cas opérationnels, ces relations peuvent justement avoir un sens uniquement pour les instances. Aucune règle de validation n'y est donc associée.	
Collaboration	*Involved Roles*	Multiplicité incohérente	Levée quand le nombre de Rôles Instances contenus est incompatible avec la multiplicité définie au niveau des Types.
		Rôle incohérent	Levée quand un Rôle instance est contenu dans une Collaboration instance mais n'est compatible avec aucun des Rôles contenus dans la Collaboration type.
	Role Player	Allocation incohérente	Levée quand l'Entité qui joue un Rôle n'est pas capable de jouer ce Rôle.

Tableau 8 – Erreurs de validation vérifiant la cohérence d'un déploiement

Annexe 3 – Règles de cohérence pour la cohérence d'un déploiement : exemples pour les entités d'entreprise

Concept	Propriété	Action de déclenchement	Contenu de la règle
Aggregate	*Aggregated Entities*	Ajout	Pour toutes les Instances de l'Agrégat, ajout d'une nouvelle instance de l'Entité agrégée.
		Suppression de l'Entité agrégée	Sans effet - Suppression des toutes les Instances de l'Entité agrégée géré par la propriété PropagatesDelete de la relation IsInstanceOf au niveau de la classe Prototype.
		Détachement de l'Entité agrégée[28]	Pour toutes les Instances de l'Agrégat, détachement de toutes les Instances de l'Entité agrégée. Ces Instances ne sont pas supprimées du modèle.
		Modification de la cardinalité	Pour chaque instance de l'Agrégat : - Si le nombre d'Entités agrégées Instances de l'Entité concernée est compatible avec la nouvelle cardinalité, sans effet - Si le nombre d'Entités agrégées Instances de l'Entité concernée est inférieur à la nouvelle cardinalité min., ajout d'autant de nouvelles Instances de l'Entité agrégée que nécessaire pour atteindre ce min. - Si le nombre d'Entités agrégées Instances de l'Entité concernée est supérieur à la nouvelle cardinalité max., sans effet (une règle de validation sera levée au niveau de l'instance d'Agrégat)
COI	*Gathered Entities*	Ajout	Sans effet – Une règle de validation sera levée au niveau de l'instance de COI
		Suppression de l'Entité agrégée	Sans effet - Suppression des toutes les Instances de l'Entité agrégée géré par la propriété PropagatesDelete de la relation IsInstanceOf au niveau de la classe Prototype.
		Détachement de l'Entité participante[29]	Sans effet – Une erreur de validation sera (si besoin) levée au niveau de l'instance de COI
		Modification de la cardinalité	

[28] Correspond à la suppression de la relation entre éléments sans suppression de l'élément cible.
[29] Correspond à la suppression de la relation entre éléments sans suppression de l'élément cible.

Physical Enterprise Entity	*Contained Entities*	Ajout	Pour toutes les Instances de l'Entité Physique, ajout d'une nouvelle instance de l'Entité contenue.
		Suppression de l'Entité contenue	Sans effet - Suppression des toutes les Instances de l'Entité contenue géré par la propriété PropagatesDelete de la relation IsInstanceOf au niveau de la classe Prototype.
		Détachement de l'Entité contenue[30]	Pour toutes les Instances de l'Entité Physique, détachement de toutes les Instances de l'Entité contenue. Ces Instances ne sont pas supprimées du modèle.
		Modification de la cardinalité	Pour chaque instance de l'Entité Physique : - Si le nombre d'Entités contenues Instances de l'Entité concernée est compatible avec la nouvelle cardinalité, sans effet - Si le nombre d'Entités contenues Instances de l'Entité concernée est inférieur à la nouvelle cardinalité min., ajout d'autant de nouvelles Instances de l'Entité agrégée que nécessaire pour atteindre ce min. - Si le nombre d'Entités contenues Instances de l'Entité concernée est supérieur à la nouvelle cardinalité max., sans effet (une règle de validation sera levée au niveau de l'instance d'Entité Physique)
	Hosted Entities	Ajout	Sans effet – La définition des relations Hosts au niveau type ne doit pas être une restriction pour le niveau instance.
		Suppression de l'Entité hébergée	Sans effet - Suppression des toutes les Instances de l'Entité agrégée géré par la propriété PropagatesDelete de la relation IsInstanceOf au niveau de la classe Prototype.
		Détachement de l'Entité hébergée[31]	Sans effet – Une erreur de validation sera (si besoin) levée au niveau de l'instance d'Entité Physique.
		Modification de la cardinalité	Pour chaque instance de l'Entité hôte, dans chaque instance de l'Entité contexte de la relation : - Si le nombre d'Entités hébergées Instances de l'Entité concernée est compatible avec la nouvelle cardinalité, sans effet - Si le nombre d'Entités hébergées Instances de l'Entité concernée est inférieur à la nouvelle cardinalité min., on crée autant de relations d'hébergement possibles jusqu'à atteindre ce min. Cependant, même si le contexte n'a pas assez de descendants pour permettre de créer autant de relations que le préconise la cardinalité, aucune entité n'est instanciée. - Si le nombre d'Entités hébergées Instances de l'Entité concernée est supérieur à la nouvelle cardinalité max., sans effet (une règle de validation sera levée au niveau de l'instance d'Entité hôte)

Tableau 9 – Règles de cohérences assurant la cohérence d'un déploiement en ce qui concerne les entités d'entreprise

[30] Correspond à la suppression de la relation entre éléments sans suppression de l'élément cible.
[31] Correspond à la suppression de la relation entre éléments sans suppression de l'élément cible.

Annexe 4 – Exemple d'interface d'IDEA Designer

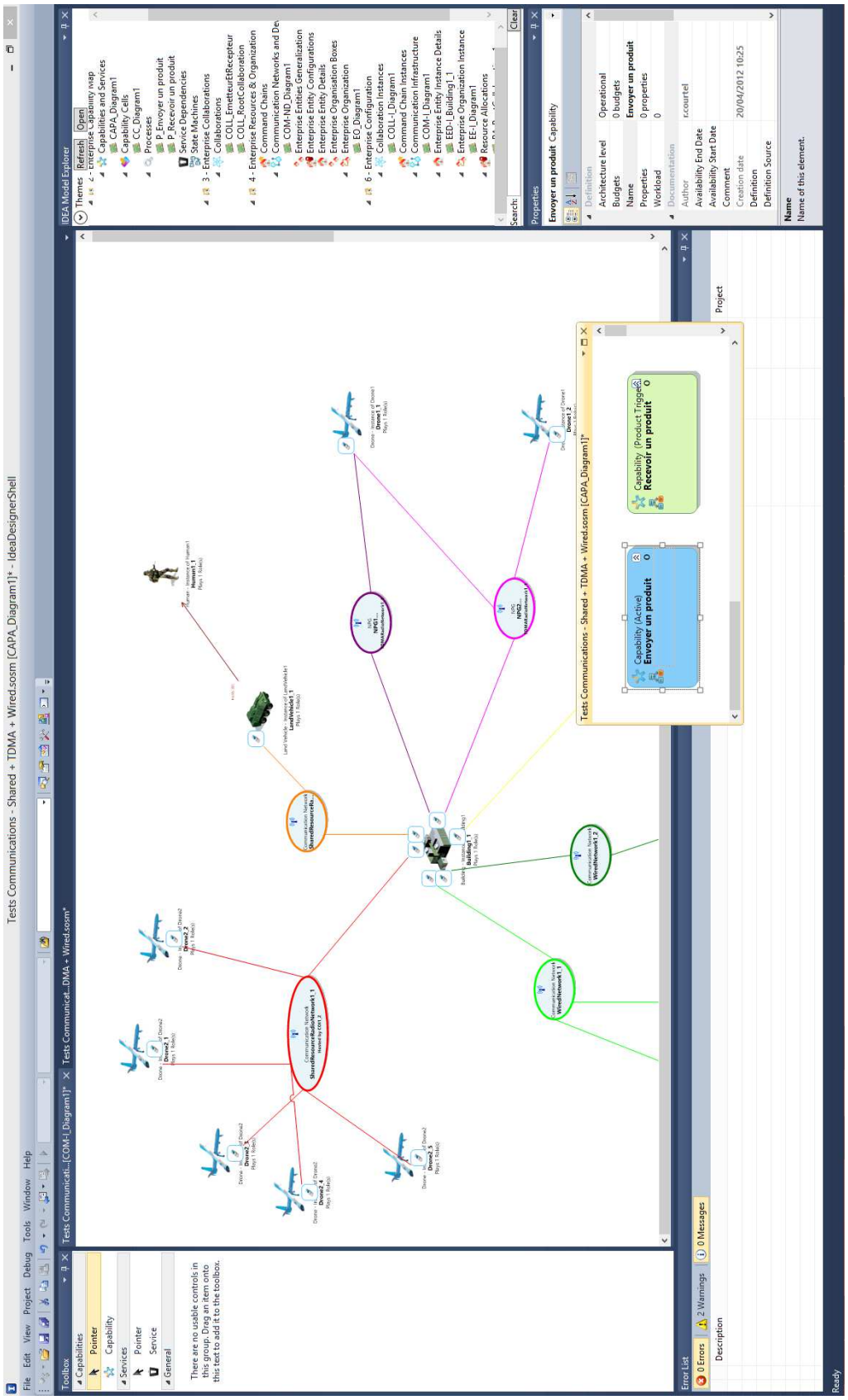


Figure 46 – Exemple d'interface d'IDEA Designer

Annexe 5 – Exemples d'interfaces d'IDEA Performer

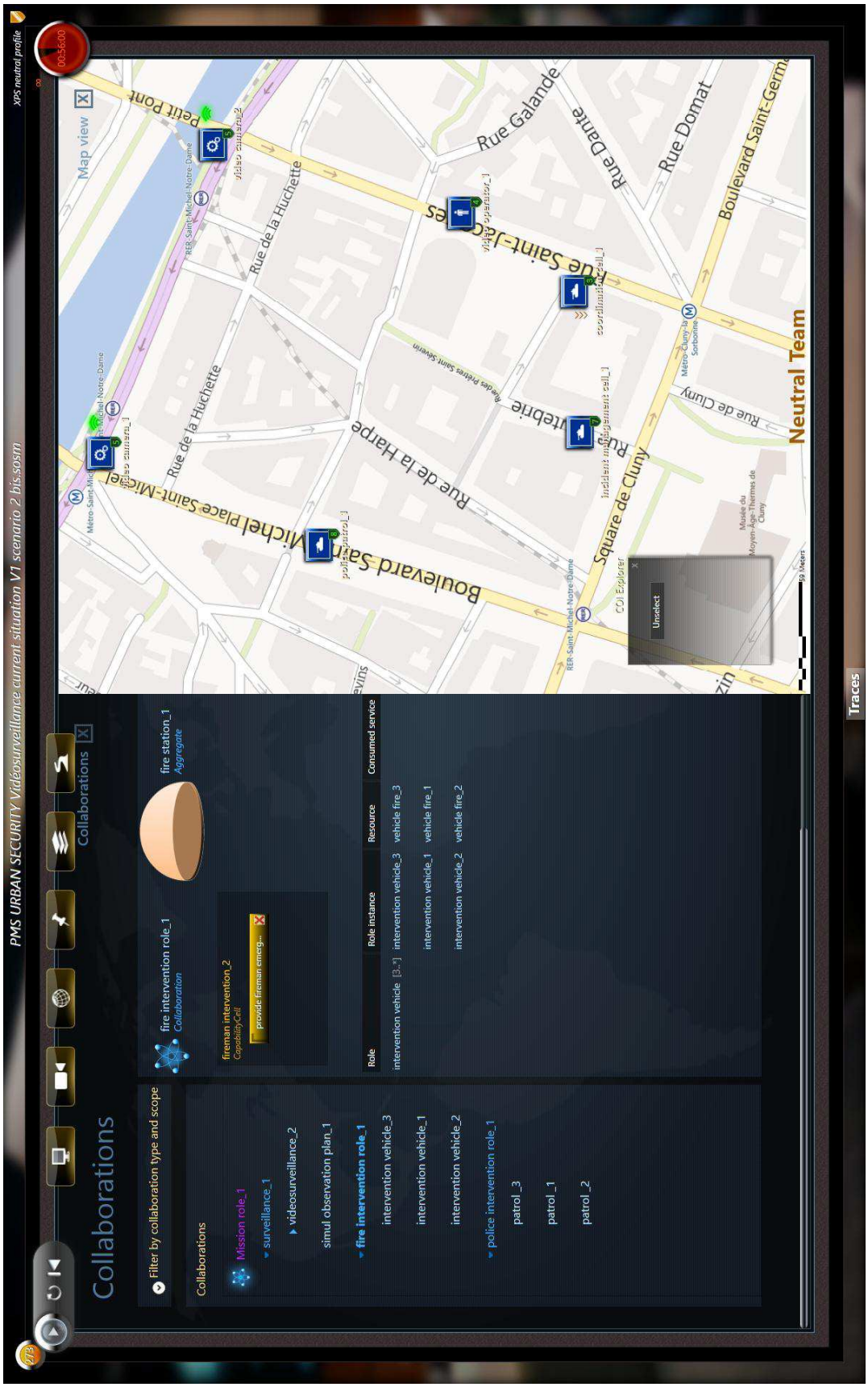


Figure 47 – Exemple d'interface d'IDEA Performer (Cartographie et entités d'entreprise)

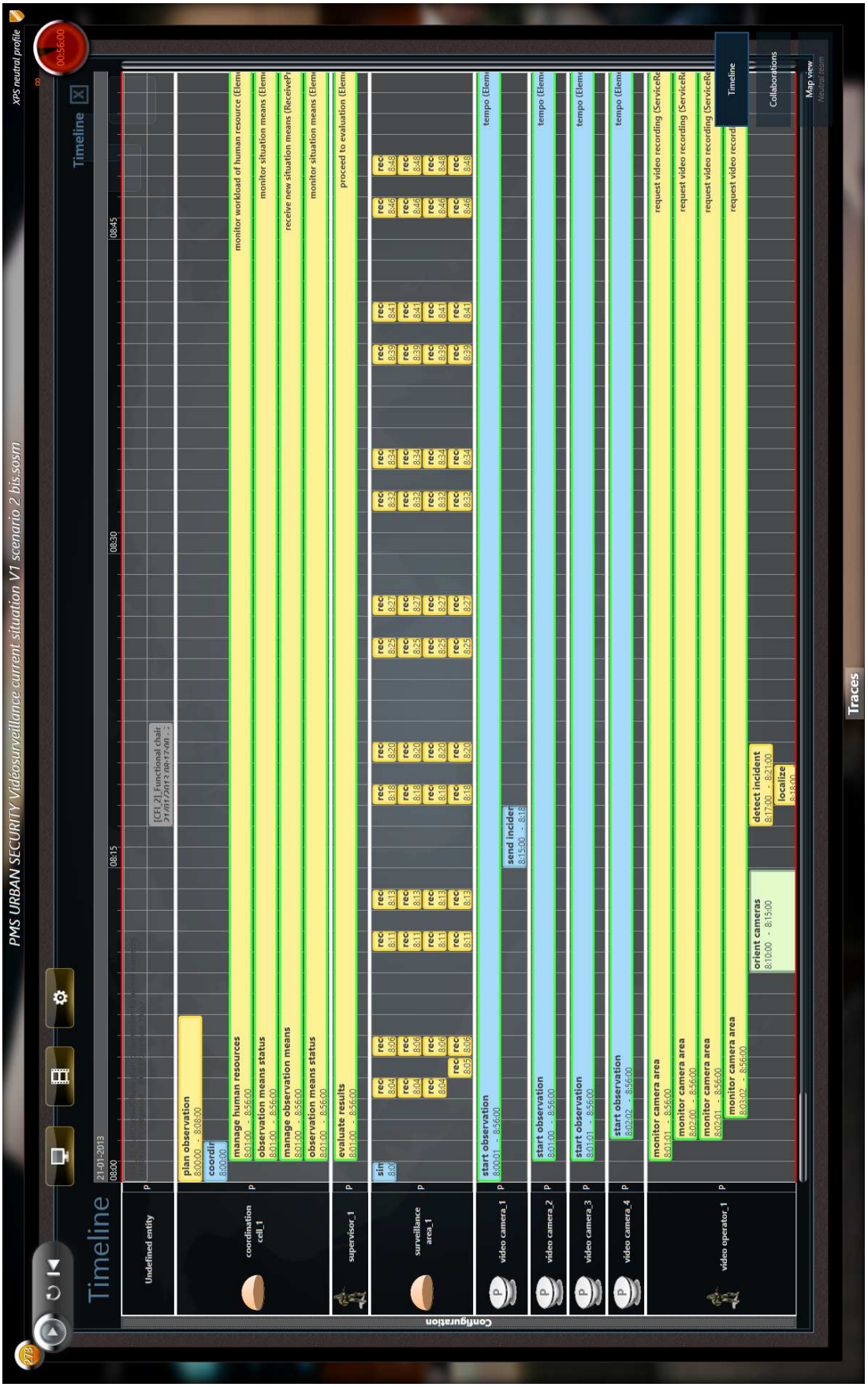


Figure 48 – Exemple d'interface d'IDEA Performer (Supervision de l'exécution des processus)

Annexe 6 – Diagrammes de modélisation d'IDEA Designer

Le tableau suivant présente l'intégralité des diagrammes de modélisation disponibles dans IDEA Designer. Les diagrammes dont le contenu n'est pas adressé en détail dans cette thèse sont présentés dans des cellules grisées.

Nom	Description
Point de vue COMMONS	
Exceptions	Description des exceptions métier
Information & Datamodel	Description des produits informationnels
Products	Description des produits physiques
Point de vue ENTERPRISE CAPABILITY MAP	
Capabilities & Services	Décomposition et classification des capacités et des services
Capability Cells	Regroupement des capacités et des services en cellules
Processes	Description du processus associé à une capacité (un diagramme par capacité)
State Machines	Description de la machine à états orchestrant une cellule capacitaire (un diagramme par cellule capacitaire)
Point de vue ENTERPRISE COLLABORATIONS	
Collaborations	Décomposition des collaborations (un diagramme par collaboration)
Point de vue ENTERPRISE RESOURCES & ORGANIZATION	
Command Chains	Définition des chaînes de commandement
Communication Networks & Devices	Définition des réseaux de communications et des entités d'entreprise représentant les équipements de communication
Enterprise Entity Configurations	Spécification des configurations possibles pour une entité d'entreprise
Enterprise Entity Details	Décomposition d'une entité d'entreprise (un diagramme par entité d'entreprise)
Enterprise Organization	Organisation des entités d'entreprise
Point de vue ENTERPRISE CONFIGURATION	
Collaboration Instances	Vue d'ensemble des collaborations de niveau instance domaine
Command Chain Instances	Vue d'ensemble des chaînes de commandement de niveau instance domaine
Communication Infrastructure	Définition de l'infrastructure de communication
Enterprise Organization Instances	Organisation des entités d'entreprise de niveau instance domaine
Enterprise Entity Instances Details	Décomposition d'une entité d'entreprise de niveau instance domaine (un diagramme par entité d'entreprise)
Resource Allocation	Allocation des entités d'entreprises de niveau instance domaine aux rôles de niveau instance domaine

Tableau 10 – Diagrammes de modélisation d'IDEA Designer

Annexe 7 – Etude de correspondance DSL IDEA - NAF

Le niveau de granularité et de généricité du langage IDEA a été voulu proche du NAF. Il convient toutefois de rappeler que si le but premier des termes et des concepts d'architecture introduits par ce métamodèle est de capturer de façon fidèle dans des modèles les constituants d'une architecture, la principale exploitation visée des modèles correspondants est la simulation de l'architecture par le bais de leur exécution. Dans le cas du NAF, la principale exploitation visée des modèles est en revanche la confrontation des points de vue et le partage de connaissance entre les architectes et avec les autres acteurs de l'ingénierie.

Bien qu'il reprenne certains termes et concepts du NAF, le DSL IDEA n'a ainsi pas été conçu pour être conforme à la spécification NAF mais pour s'orienter vers la simulation. Le NAF restant cependant un standard pour une grande partie des études d'architecture d'entreprise ou de Systèmes de Systèmes, aussi il importe pour la validation du langage IDEA d'identifier précisément les recouvrements avec cet Architecture Framework. Pour ce faire, nous avons cherché à produire chacun des différents produits spécifiés par le NAF à partir d'un modèle IDEA.

A la définition du périmètre du métamodèle IDEA ont été volontairement écartées les informations adressés par certaines vues et certains produits NAF. Une partie de ces vues et produits a été écartée car les informations qu'elles contiennent ont un but documentaire, et ne représentent pas des constituants d'architecture. C'est le cas par exemple de la vue « All Views », qui contient des informations d'ordre général sur le modèle comme un résumé ou des définitions. Les autres ont été écartées car les préoccupations qu'elles adressent relèvent d'une phase d'ingénierie ultérieure à la phase amont de conception de l'entreprise, dans laquelle se place une approche de prototypage rapide. C'est le cas par exemple de la vue « Programme Views », qui établit les projets formalisant la conception des constituants de l'entreprise. Le choix d'écarter des modèles IDEA le périmètre couvert par ces différents produits a été accepté et validé par les utilisateurs, dans la mesure où ils n'ont effectivement pas éprouvé le besoin de représenter les informations correspondantes en phase amont de conception de l'architecture.

En ce qui concerne le périmètre de NAF initialement retenu pour être également couvert par le DSL IDEA, la création d'un produit donné à partir d'un modèle IDEA fait apparaître plusieurs niveaux de couverture distincts :

- **Niveau 1** – Le DSL IDEA couvre **intégralement** l'ensemble des concepts nécessaires à la création du produit. Dans certains cas, IDEA Designer propose de plus un diagramme de modélisation dont le périmètre est équivalent à celui du produit NAF.

  Par exemple, le diagramme IDEA 'Capabilities & Services' (point de vue CAPABILITY MAP) expose une classification des capacités de l'entreprise correspondant au produit NCV-2 ('Capability Taxonomy').
- **Niveau 2** – La création du produit nécessite une **transformation de modèle** pour faire apparaître un concept NAF n'ayant pas d'équivalent direct parmi les concepts du métamodèle IDEA.

  C'est en particulier le cas de la notion NAF de *Needline*, exprimant une dépendance entre deux nœuds opérationnels dans le produits NOV-2 ('Operational Node Connectivity Description').

Bien que le langage IDEA ne contienne pas de concept strictement équivalent, il permet de décrire l'information correspondance : pour deux nœuds opérationnels donnés, correspondant dans le modèle IDEA à deux entités d'entreprise de niveau instance domaine, les *Needline* peuvent être déduites par le biais des interactions établies entre les rôles joués par ces deux entités.

- **Niveau 3** – Le DSL IDEA **ne permet pas** de modéliser les éléments nécessaires pour créer le produit, mais les informations requises peuvent être déduites soit d'une simulation, soit par le biais du mécanisme d'extensibilité des outils IDEA. Ce cas reste cependant anecdotique, dans la mesure où il ne concerne que trois produits.

Le tableau suivant présente la correspondance entre les produits NAF v3 et le DSL IDEA, indiquée de la manière suivante :

- **Couvert** – Périmètre de la vue pris en compte par le DSL IDEA. L'information correspondante peut être un sous-ensemble d'un diagramme de modélisation, être équivalente à un diagramme de modélisation ou être répartie entre plusieurs diagrammes de modélisation. (cellules bleues)
- **Non couvert** – Périmètre de la vue non pris en compte par le DSL IDEA. (cellules grisées)

N°	Nom	Correspondance IDEA
All Views		
NAV-1	Overview and Summary Information	**Non couverts** (Décrivent des informations générales sur le modèle d'architecture, et non des constituants de l'architecture)
NAV-2	Integrated Dictionnary	
NAV-3	Metadata	
Programme Views		
NPV-1	Programme Portfolio Relationships	**Non couverts** (Décrivent des informations générales sur le modèle d'architecture, et non des constituants de l'architecture)
NPV-2	Programme to Capability Mapping	
Capability Views		
NCV-1	Capablity Vision	**Non couvert** (Décrit les motivations et objectifs de l'entreprise, et non des constituants de l'architecture)
NCV-2	Capability Taxonomy	**Couvert** (Diagramme 'Capabilities & Services')
NCV-3	Capability Phasing	**Non couvert** (Décrit les aspects temporels de l'entreprise, et non des constituants de l'architecture)
NCV-4	Capability Dependencies	**Couvert** (Diagrammes 'Collaborations' et 'Capabilities & Services '
NCV-5	Capability to Organisational Activity Mapping	**Couvert** (Diagrammes 'Collaboration Instances', 'Enterprise Organization Instance' et 'Capabilities & Services')
NCV-6	Capability to Operational Activity Mapping	**Couvert** (Diagrammes 'Capabilities & Services' et 'Process')
NCV-7	Capability to Services Mapping	**Couvert** (Diagramme 'Capabilities & Services')
Operational Views		
NOV-1	High Level Operational Concept Graphic	**Non couvert** (« vue d'artiste »)

NOV-2	Operational Node Connectivity Description	**Couvert** (Diagrammes 'Collaboration Instances' et 'Enterprise Organization Instance')
NOV-3	Operational Information Exchange Matrix	**Couvert** (Diagrammes 'Products', 'Information & Datamodel' et 'Collaboration Instances')
NOV-4	Organizational Relationships Chart	**Couvert** (Diagramme 'Collaboration Instances' et 'Command Chain Instances')
NOV-5	Operational Activity Model	**Couvert** (Diagrammes 'Process')
NOV-6a	Operational Rule Model	**Couvert** (Par la possibilité d'associer des fonctions d'extension aux activités de processus)
NOV-6b	Operational State Transition Description	**Couvert** (Diagrammes 'State Machines')
NOV-6c	Operational Event-Trace Description	**Couvert** (Par la possibilité de produire des diagrammes de séquence correspondant à une exécution particulière. IDEA n'adresse cependant pas la modélisation de diagrammes de séquence prescriptifs)
NOV-7	Information Model	**Couvert** (Diagrammes 'Products' et 'Information & Datamodel')
Service Views		
NSOV-1	Services Taxonomy	**Couvert** (Diagramme 'Capabilities & Services')
NSOV-2	Services Definition (Services Properties)	**Couvert** (Diagramme 'Capabilities & Services')
NSOV-3	Services to Operational Activities Mapping Note : en NAF v3.1, devient Capability to Service Mapping	**Couvert** (Diagramme 'Capabilities & Services')
NSOV-4	Services Orchestration Note : en NAF v3.1, cette vue est éclatée en trois vues et change sensiblement de sens	**Couvert** (Diagrammes 'Collaborations')
NSOV-5	Service Behaviour Note : en NAF v3.1, devient Service Functionality	**Couvert** (Diagramme 'Capabilities & Services' et 'Process')
System Views		
NSV-1	Systems Interface Description	**Couvert** (Diagramme 'Enterprise Organization Instances' et 'Communication Infrastructure')
NSV-2a	System Port Specification	**Non couvert** (Niveau de détail < prototypage d'architecture)
NSV-2b	System to System Port Connectivity	
NSV-2c	System Connectivity Clusters	
NSV-3	Systems to Systems Matrix Note : en NAF v3.1, devient Resource Interaction Matrix	**Couvert** (Diagramme 'Enterprise Organization Instances' et 'Communication Infrastructure')

NSV-4	Systems Functionality Description	**Couvert** (Diagrammes 'Process')
NSV-5	Systems Function to Operational Activity Traceability Matrix	**Couvert** (Diagramme 'Enterprise Organization Instances' et 'Collaboration Instances')
NSV-6	Systems Data Exchange Matrix	**Couvert** (Diagrammes 'Products', 'Information & Datamodel' et 'Collaboration Instances')
NSV-7	Systems Quality Requirements description	**Non couvert** (Niveau de détail < prototypage d'architecture)
NSV-8	Systems Evolution Description Note : en NAF v3.1, devient System Configuration Management	
NSV-9	Systems Technology Forecast	**Non couvert** (Niveau de détail < prototypage d'architecture)
NSV-10a	Systems Rules Mode	**Couvert** (Par la possibilité d'associer des fonctions d'extension aux activités de processus)
NSV-10b	Systems State Transition Description	**Couvert** (Diagrammes 'State Machines')
NSV-10c	Systems Event-Trace Description	**Couvert** (Par la possibilité de produire des diagrammes de séquence correspondant à une exécution particulière. IDEA n'adresse cependant pas la modélisation de diagrammes de séquence prescriptifs)
NSV-11a	Logical Data Model	**Couvert** (Diagramme 'Information & Datamodel')
NSV-11b	Physical Data Model	**Couvert** (Diagramme 'Products')
NSV-12	Systems Service Provision	**Couvert** (Diagramme 'Enterprise Organization Instances' et 'Collaboration Instances')
Technical Views		
NTV-1	Technical Standards Profile	**Non couverts** (Niveau de détail < prototypage d'architecture)
NTV-2	Technical Standards Forecast	
NTV-3	Standard Configurations	

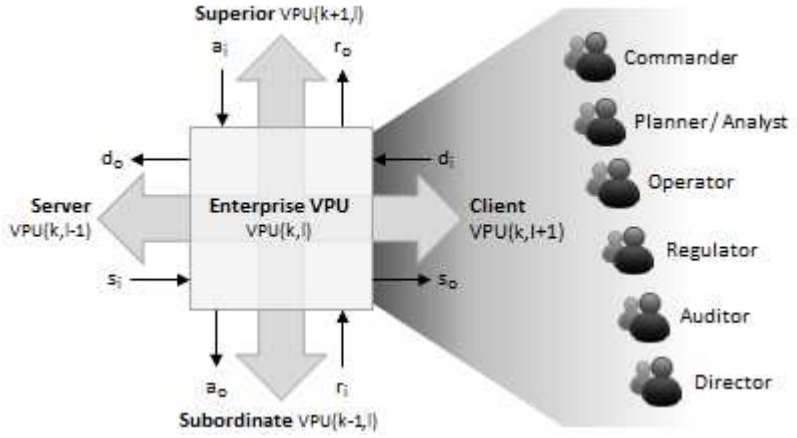
Tableau 11 – Correspondanc DSL IDEA-NAF

Annexe 8 – Etude de correspondance DSL IDEA – Modèles de Bayne

Les travaux de Bayne ont pour objectif de décrire une structure type pour les entreprises afin de permettre l'évaluation de leur performance par l'intermédiaire de modèles mathématiques. Contrairement aux études et réalisations effectuées dans le cadre de cette thèse et des outils IDEA, les travaux de Bayne portent donc sur des *modèles* et non sur un *métamodèle*. Les modèles de Bayne représentent toutefois une description générique d'une entreprise, adaptable dans tous les domaines métier, et répandue et acceptée dans la communauté du Command & Control à laquelle s'adresse en partie le langage IDEA. Il nous a par conséquent paru fondamental d'évaluer le lien entre ce dernier et les modèles de Bayne, en cherchant à modéliser les composantes d'une entreprise selon Bayne dans le formalisme IDEA.

Comme évoqué précédemment, les modèles de Bayne sont basés sur un concept de VPU (Value Production Unit), et en décrivent deux aspects notoires : l'organisation d'un VPU au sein de la fédération, et son comportement. Dans la suite de cette partie, nous allons étudier les possibilités de réaliser un modèle IDEA d'un VPU selon ces deux aspects.

Considérons pour commencer l'organisation d'un VPU au sein de la fédération, telle qu'introduite dans la Figure 2 rappelée ci-après.



[Rappel] Figure 2 – Organisation d'un VPU

- **Modélisation de la structure interne d'un VPU** – Les acteurs présentés dans les modèles de Bayne sont d'ordre fonctionnel, définissant un domaine de responsabilité mais pas les ressources auxquelles sont allouées les responsabilités. Qu'ils soient VPUs ou membres de l'équipe d'un VPU, ces acteurs sont par ailleurs toujours décomposables en sous-acteurs.

Dans le langage IDEA, il s'agit là de caractéristiques similaires à celles du concept *Role*. Les VPU seront donc modélisés sous forme de Roles et les membres de l'équipe d'un VPU comme les rôles impliqués dans la collaboration correspondante, comme illustré sur la Figure 49.

Figure 49 - Modélisation de l'organisation d'un VPU dans le langage IDEA : implémentation du rôle « VPU k,l »

- **Interactions selon l'axe hiérarchique** – L'axe hiérarchique selon Bayne (noté « k ») décrit des relations de commandement, qui se traduisent par la mise en œuvre d'échanges d'information entre un VPU, son supérieur et son subordonné.
  Dans le langage IDEA, ces échanges d'information sont modélisés au niveau des collaborations par des interactions mettant en œuvre des informations de type ordre / compte rendu. Dans le modèle IDEA d'un VPU, les interactions d'un VPU avec son supérieur et son (ses) subordonné(s) par le biais de *Endpoints* d'envoi et de réception d'informations ordre / compte-rendu.
- **Interactions selon l'axe logistique** – L'axe logistique selon Bayne (noté « l ») décrit des relations de type producteur / consommateur.
  Dans le langage IDEA, cela se traduit de manière relativement naturelle par des interactions de service : le VPU consomme un service depuis son VPU fournisseur (*Endpoint* de consommation du service « VPU k,l-1 service » sur la Figure 50), et fournit un service à son consommateur (*Endpoint* de fourniture du service « VPU k,l service »). Les données échangées selon cet axe entre les VPUs dans le modèle de Bayne seront modélisées dans des échanges internes aux services.

La Figure 50 ci-après présente le modèle IDEA des interactions entre un VPU noté « k, l » et ses voisins (supérieur, subordonné, fournisseur, consommateur).

Le modèle d'organisation d'un VPU réalisé ainsi dans IDEA décrit une collaboration comprend séparément l'implémentation d'une collaboration représentant le VPU (cf. Figure 49), et les interactions réalisées par cette collaboration avec ses voisins (cf. Figure 50). Le lien entre les deux, c'est-à-dire la manière selon laquelle les rôles collaborent pour contribuer aux échanges du VPU avec ses voisins, n'est pas précisé dans le modèle de Bayne. Les rôles y interagissent par l'intermédiaire de leurs actions sur le processus de production, et c'est également ce dernier qui définit les modalités d'échanges du VPU avec les VPUs voisins. Le contenu de ce processus est cependant spécifique à l'entreprise considérée, et ne peut donc pas être modélisé de manière générique. Dans le cadre d'étude de cette partie, c'est-à-dire la traduction d'un modèle de Bayne dans le langage IDEA, le modèle ainsi produit ne pourra pas comprendre de liens entre les rôles et vers l'interface du VPU.

S'il est possible de représenter les concepts des modèles d'organisation d'entreprise selon Bayne dans le langage proposé dans le cadre de cette thèse, on constate que le modèle IDEA résultant est incomplet. Ce non recouvrement n'est cependant pas surprenant, car le périmètre d'étude et les

objectifs des modèles de Bayne et celui du langage IDEA ne sont pas similaires. Les modèles de Bayne ont pour objectif la définition d'un formalisme générique pour l'optimisation par les mathématiques des processus de C2, alors que les modèles IDEA sont conçus pour représenter et expérimenter l'architecture d'une entreprise. Les modèles de Bayne ayant vocation à être appliqués à toute entreprise, le modèle IDEA élaboré dans cette partie peut toutefois être considéré comme un *patron* de modélisation. Ce patron propose alors une base de modèle claire et générique, dont peut être dérivé un modèle spécifique pour une entreprise en particulier :

- Une structure unique d'organisation interne pour une collaboration (Figure 49), en l'occurrence les différents rôles génériques identifiés comme contributeurs à un VPU ;
- Un patron d'interactions entre collaborations (Figure 50), basé sur une séparation claire entre des échanges d'informations selon un axe hiérarchique et des échanges de service selon un axe logistique) ;

Pour dériver de ce patron un modèle IDEA spécifique à une entreprise, complet et simulable, il faut notamment prendre en compte les points suivants :

- Identifier l'organisation d'acteurs venant prendre en charge les VPUs (c.-à-d. jouer les rôles correspondants) ;
- Spécifier les processus de production et de C2 des différents VPUs. La simulation d'un modèle IDEA étant basé sur les processus, la définition de ces derniers est indispensable pour obtenir un modèle complet. Les modèles de Bayne s'intéressant à l'étude du Command & Control (C2) de l'entreprise, leur intérêt réside dans la manière dont les processus de C2 interagissent avec le processus de production et non dans le détail des différents processus.

L'intérêt de mettre en œuvre d'un tel patron est alors de permettre une transition fluide entre la démarche d'architecture dans laquelle s'inscrit la création d'un modèle IDEA, et la démarche d'étude et d'optimisation du Command & Control concernée par les modèles de Bayne. En effet, un modèle d'architecture IDEA basé sur ce patron pourra être aisément transcrit vers le formalisme proposé par Bayne, puis faire l'objet des analyses mathématiques mises en place pour ce formalisme. Dans le cadre d'une démarche de conception d'entreprise, l'analyse du modèle résultant pourra alors apporter une contribution importante en termes d'évaluation des aspects de C2, tout en conservant une traçabilité forte vis-à-vis de la phase d'architecture.

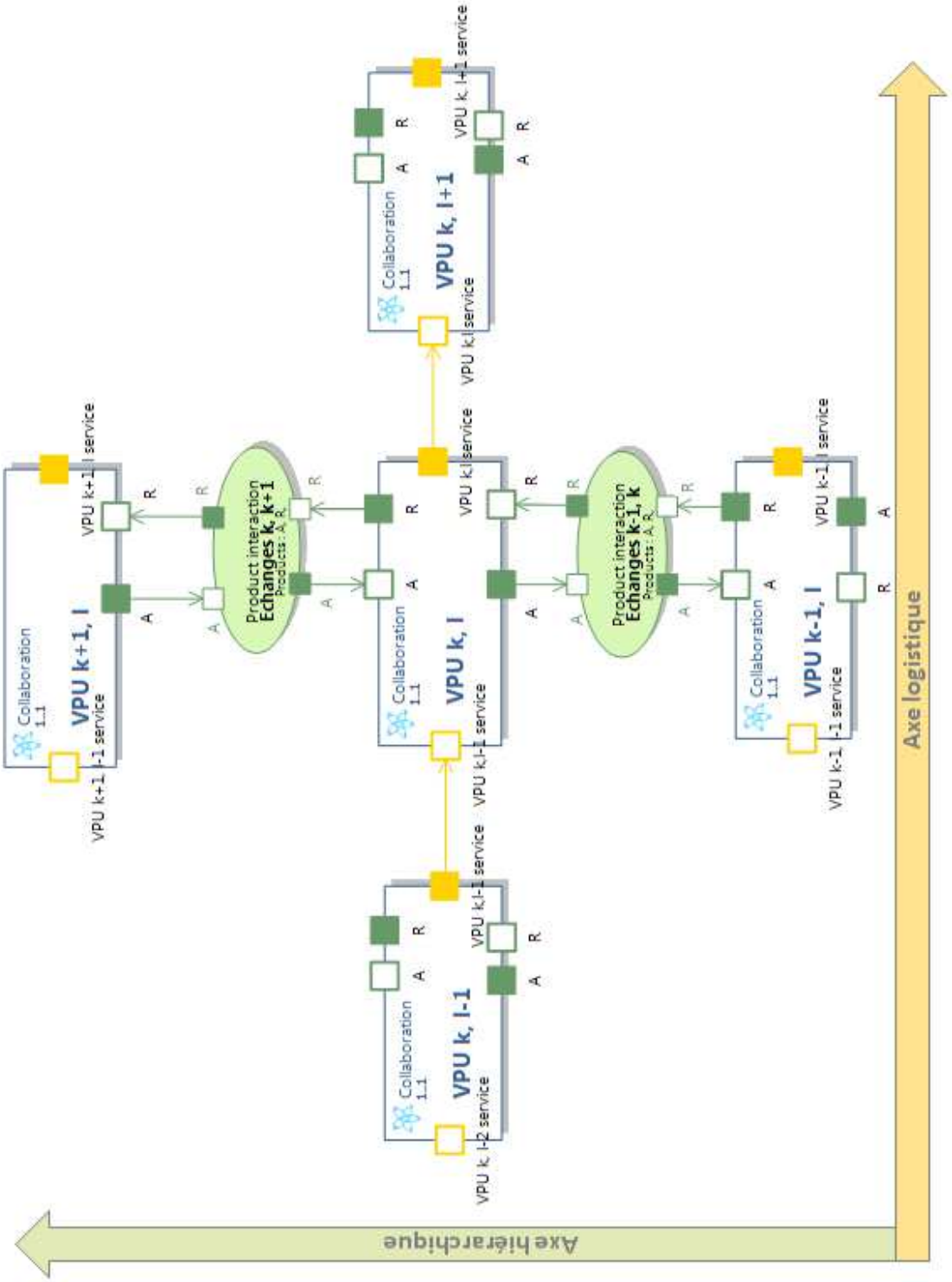


Figure 50 – Modélisation de l'organisation d'un VPU dans le langage IDEA : interactions du rôle « VPU k,l » dans la fédération

Références

[1] MAIER, M. Architecting Principles for System of Systems. In : INCOSE Systems Engineering Journal, 1998, Vol.1, No.4, pp. 267-284

[2] LUZEAUX, D., RUAULT, J.R. Ingénierie des systèmes de systèmes - concepts et illustrations pratiques. Hermes Science, 2008, 225p. ISBN 978-2-7462-1875-8

[3] BAYNE, J. Creating Rational Organizations : Theory of Enterprise Command and Control. MCSI, 2006, 262p. ISBN 097885960X

[4] BAYNE, J., DIGGS, D. C2 in the Joint Task Force (JTF) Enterprise. In : 11th International Command and Control Research and Technology Symposium (ICCRTS), 2006, San Diego, USA

[5] BAYNE, J., PAUL, R. Policy-based Command and Control. In : 10th International Command and Control Research and Technology Symposium (ICCRTS), 2005, McLean, USA

[6] DoD (US Department of Defense). DoD Architecture Framework Version 2.02 [en ligne]. Disponible sur : http://dodcio.defense.gov/sites/dodaf20/index.html (consulté le 27/02/2012)

[7] NATO (North Atlantic Treaty Organisation). Architecture Home Page [en ligne]. Disponible sur : http://www.nhqc3s.nato.int/HomePage.asp?session=420232627 (consulté le 27/02/2012)

[8] DoD (US Department of Defense). DoD Dictionary of Military Terms [en ligne]. Disponible sur : http://www.dtic.mil/doctrine/dod_dictionary/ (consulté le 27/02/2012)

[9] International Council on Systems Engineering (INCOSE). Systems Engineering Vision 2020, Technical Report INCOSE-TP-2004-004-02, 2007.

[10] OMG (Object Management Group). Meta-Object Facility [en ligne]. Disponible sur : http://www.omg.org/mof/ (consulté le 01/03/2012)

[11] OMG (Object Management Group). Unified Modeling Language [en ligne]. Disponible sur : http://www.uml.org/ (consulté le 01/03/2012)

[12] AUDIBERT, L. UML 2 [en ligne]. Disponible sur http://laurent-audibert.developpez.com/Cours-UML/ (consulté le 01/03/2012)

[13] OMG (Object Management Group). OMG Systems Modeling Language [en ligne]. Disponible sur : http://www.omgsysml.org/ (consulté le 01/03/2012)

[14] LUDWIG, M., FARCET, N. Evaluating Enterprise Architectures through Executable Models. In : 15th International Command and Control Research and Technology Symposium (ICCRTS), 2010, Santa Monica, USA

[15] LUDWIG, M., FARCET, N., DUMOND, R. Executable Systems of Systems Architecture Models. In : Modelling and Simulation of Complex Systems in Public Safety and Military Operations. Meeting Proceedings RTO-MP-MSG-082. Neuilly-sur-Seine, France : RTO, 2010, pp. 17-1 – 17-2

[16] DIXSON, M., GENIK, L. Application of Command and Control Architecture Frameworks for Security and Emergency Management Planning. In : Modelling and Simulation of Complex

Systems in Public Safety and Military Operations. Meeting Proceedings RTO-MP-MSG-082. Neuilly-sur-Seine, France : RTO, 2010, pp. 16-1 – 16-12

[17] BAGNULO, A., HARVEY, G. Executable Architecture – Survey and Suitability Assessment Study of Systems Engineering Standards and Methodologies. DRDC-VALCARTIER-CR-2008-180 [en ligne]. Valcartier, Canada : Defence R&D Canada, 2008, 92p. Disponible sur : http://cradpdf.drdc-rddc.gc.ca/PDFS/unc87/p531003.pdf (consulté le 24/02/2012)

[18] IRISA Rennes. Kermeta Home Page [en ligne]. Disponible sur : http://www.kermeta.org/ (consulté le 01/03/2012)

[19] PAWLOWSKI, T., BARR, P., RING, S.J. Applying Executable Architectures to Support Dynamic Analysis of C2 Systems. In : 2004 Command and Control Research and Technology Symposium (CCRTS), 2004, San Diego, USA

[20] OMG (Object Management Group). UML 2.4.1 Infrastructure Specification. OMG Document n°formal/2011-80-05 [en ligne]. OMG, 2011, 230p. Disponible sur : http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/ (consulté le 05/03/2012)

[21] HILLIARD, R. On Representing Variation. In : 4th European Conference on Software Architecture (ECSA'2010), 2010, Copenhague, DK.

[22] MORIN, B., PERROUIN, G., LAHIRE, P., BARAIS, O. VANWORMHOUDT, G., JEZEQUEL, J-M. Weaving Variability into Domain Metamodels. In : 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09), 2009, Denver, USA

[23] HAUGEN, Ø., MØLLER-PEDERSEN, B., OLDEVIL, J., OLSEN, G. K., SVENSEN, A. Adding Standardized Variability to Domain Specific Languages. In : 12th International Software Product Line Conference (SPLC), 2008, Limerick, IRL

[24] FLEUREY, F. HAUGEN, Ø., MØLLER-PEDERSEN, B., OLDEVIL, J., OLSEN, G. K., SVENSEN, A, ZHANG, X. A Generic Language and Tool for Variability Modeling. SINTEF Report A13505, 2009

[25] Mexedge. Multiply DSL Points of Views [en ligne]. Disponible sur : http://blog.mexedge.com/2009/01/13/multiply-dsl-points-of-view/ (consulté le 15/03/2012)

[26] Mexedge. Multiply DSL Diagrams [en ligne]. Disponible sur : http://blog.mexedge.com/2011/06/15/multiply-diagrams/ (consulté le 15/03/2012)

[27] ATKINSON, C., KUHNE, T. The Essence of Multilevel Metamodeling. In : UML – The Unified Modeling Language. Modeling Languages, Concepts and Tools Lecture Notes in Computer Science, 2001, Vol. 2185, pp 19-33

[28] MALLET, F., ANDRE, C., LAGARDE, F. Un profil UML pour la modélisation multiniveau. Rapport de recherche INRIA, inria-00482727, 2010

[29] MIP (NATO Multilateral Interoperability Programme). The Joint C3 Information Exchange Data Model (JC3IEDM) Metamodel Specification. V3.1.4, 2012

[30] OMG (Object Management Group). Unified Profile for the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defence Architecture Framework (MODAF) [en ligne]. Disponible sur : http://www.omg.org/spec/UPDM/ (consulté le 15/03/2012)

[31] MELLOR, S., BALCER, J. Executable UML: A Foundation for Model-Driven Architecture. Addison-Wesley Professional, 2002, 416p. ISBN 978-0201748048

[32] IEEE Computer Society. IEEE Recommended Practice for High Level Architecture (HLA) Federation Development Execution Process (FEDEP). IEEE Report 1516.3, 2003

[33] GARDINER, J. Linking Architecture Models and Synthetic Environments – An Integrated Project Support environment. In: 7th Annual Conference on Systems Engineering Research, 2009, Loughborough, UK

[34] TOLK, A. MUGUIRA, J. Modeling & Simulation within the Model Driven Architecture. In: Interservice / Industry Training, Simulation and Education Conference (I/ITESEC), 2004, Orlando, USA

[35] KEWLEY, R.Jr., TOLK, A. A Systems Engineering Process for Development of Federated Simulations. In: Spring Simulation Multiconference (SpringSim), 2009, San Diego, USA

[36] WISNOSKY, D., VOGEL, J., RING, S. The Road to Executable Architectures. In: 9th Annual INCOSE Region II Fall Mini-Conference, 2002, San Diego, USA

[37] OMG (Object Management Group). Business Process Model and Notation (BPMN) [en ligne]. Disponible sur : http://www.bpmn.org/ (consulté le 15/03/2012)

[38] FARCET, N., BIAU, E. Les architectures executables au service de l'architecte. Présentation interne Thales, 2011

[39] FARCET, N., SLIMANI, M., GARNIER, J-L., DUMOND, R. A Model-Based Architecture to Manage Tactical Systems of Systems. In: NATO SCI-187 Symposium on Agility, Resilience and Control in Network-Enabled Capabilities (NEC), 2008, Amsterdam, NL

[40] FARCET, N., SLIMANI, M., GARNIER, J-L., LUDWIG, M., ISRAEL, M., DUMOND, R. A Model-Based Architecture for Tactical Systems of Systems Management. In: NATO RTO-MP-IST-087 RTO Information Systems Technology Panel Symposium, 2009, Stockholm, SW

[41] OASIS SOA Reference Model [en ligne]. Disponible sur : https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm (consulté le 01/02/2013)

[42] BARDOU, D. Etude des langages à prototypes, du mécanisme de délégation, et de son rapport à la notion de point de vue. Thèse Informatique. Montpellier : Université Montpellier II, 1998, 259p.

[43] FARCET, N., IDEA, Club Escadre-DirectSIM, DGA, 2011-02-02, Arcueil

Index